Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Streamování a publikování multimediálních dat v technologii Node.js

Bc. Jan Paprota

Vedoucí práce: Ing. Jiří Smítka

12. února 2015

Poděkování

Chtěl bych tímto poděkovat vedoucímu práce panu Ing. Jiřímu Smítkovi za skvělou spolupráci a cenné rady, které mi při psaní práce poskytl.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 12. února 2015 .....

Odkaz na tuto práci

Paprota, Jan. *Streamování a publikování multimediálních dat v technologii Node.js*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce si klade za cíl důkladně analyzovat způsoby přehrávání a publikování videa na internetových stránkách. Popisuje vhodné technologie prohlížečů od nejstarších až po ty nejmodernější. V dnešní době je nejčastěji používaná technologie Flash. Prostřednictvím protokolu RTMP dokáže Flash plugin přímo na stránce přehrávat video i živé přenosy nebo publikovat data z webkamery a mikrofonu. Výstupem této práce je zdokumentovaný a otestovaný RTMP server. Umožňuje návštěvníkům internetových stránek pořádat online videokonference, přehrávat video soubory nebo nahrávat jejich vlastní videa do souborů. Tyto a mnoho dalších funkcí serveru demonstrují ukázkové aplikace.

Klíčová slova Video na vyžádání, živé přenosy, publikování, RTMP, AMF, HTTP Live Streaming, Flash Video, vzdálené volání procedur, Node.js, JavaScript, Flash, HTML5, Libav, FFmpeg.

Abstract

The objective of this thesis is to deeply analyze the means of playing and publishing video on web pages. We describe appropriate web browser technologies from the oldest to the most modern ones. Nowadays the most commonly used technology is Flash. By means of RTMP protocol the Flash plug-in is able to play videos and live broadcasts or publish data from webcam and microphone. The output of this thesis is a documented and tested RTMP server which enables visitors of web pages organize on-line video-conference, replay video files or upload their own video into a file. This and many other functions of the server are demonstrated in sample applications.

Keywords Video on Demand, Live streaming, publishing, RTMP, AMF, HTTP Live Streaming, Flash Video, Remote Method Invocation, Node.js, JavaScript, Flash, HTML5, Libav, FFmpeg.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Internetové stránky obsahují texty, obrázky a obsahují také videa, dokonce spoustu videí. Dnešní uživatelé si žádají multimediální obsah. Technologie to umožňují, tak jim multimédia můžeme nabídnout. Bohužel vložit video na internetovou stránku není jednoduchý úkol. Je to vlastně docela věda, proto se administrátoři internetových stránek obrací na externí služby, nejvíce YouTube. Video snadno nahrají a sdílí, aniž by se museli zabývat technickými detaily.

Co ale dělat v případě, když YouTube nechceme použít? Co dělat, když ho dokonce nemůžeme použít? Mohli bychom se setkat s problémy 3 administrátorů:

- „Mám malý blog. Dávám na něj videa, jak se naučit hrát na saxofon. Uživatelům se to líbí, rádi má videa sdílí. Jenže natočení jednoho videa zabere spoustu času a proto bych je rád pár korunami zpoplatnil.“

- „Na stránkách naší profesní agentury bychom chtěli umožnit svým uživatelům nahrát k životopisu krátké video. Obrázek vydá za tisíc slov, video vydá za tisíc obrázků.“

- „Náš eshop již má integrovaný textový chat. Zákazníci se na nás mohou kdykoliv obrátit se svými dotazy. Chat předčil naše očekávání, dokonce bychom jej rádi rozšířili. Chtěli bychom, aby zákazník operátora viděl. Nepíše si přece s počítačem, na druhé straně sedí skutečný člověk.“

Toto jsou jen 3 příklady. Skutečný svět jich je plný. V této práci řeším konkrétní problémy – zabývám se možnostmi, jak na internetových stránkách přehrávat video, jak ho publikovat na server a jak ho následně zpracovat.

Cíle práce

Tato práce si klade za cíl vytvořit software, který bude sloužit ke 3 účelům. Dokáže přímo uvnitř internetové stránky:

1. přehrávat video a živé vysílání,
2. pořádat video konference mezi uživateli,
3. nahrávat video soubory z webkamery a mikrofonu.

Rozvržení práce

K úspěšnému splnění cílů je potřeba postupovat podle následujících bodů:

- Analyzovat technologie podporované prohlížeči.
- Prozkoumat existující softwarová řešení a bude-li to možné, využít je také jako základ práce.
- Navrhnout vhodnou strukturu aplikace s důrazem na univerzálnost.
- Software implementovat, zdokumentovat a otestovat.
- Popsat použití softwaru včetně nastavení, které má vliv na jeho výkon a kvalitu zvukových i obrazových dat.
- Prokázat splnění cílů ukázkovou aplikací.

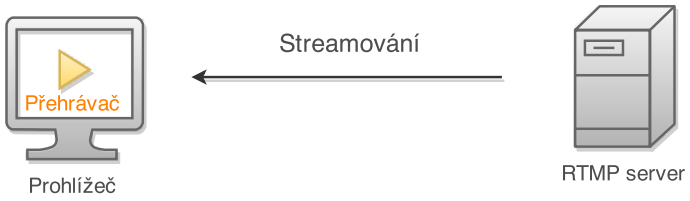
Postup vypracování popíšu v samostatné kapitole Realizace – a to zejména problémy a komplikace, se kterými se setkám.

Upřesnění pojmů

V práci hojně používám termíny streamování a publikování. Bohužel jsem nenašel v českém prostředí vhodnější pojmy, a proto jsem musel jejich definici částečně upravit. Nepochybně bych měl hned v úvodu čtenáře seznámit se zamýšlenou definicí:

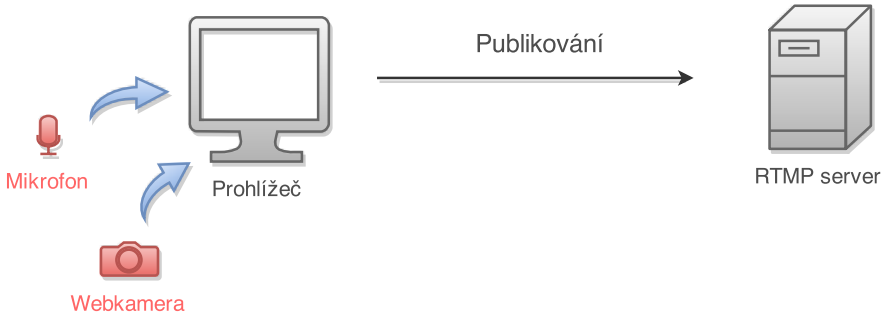
Streamování dat Streamování, anglicky *streaming*, označuje přenos multimediálních dat mezi dvěma síťovými uzly. V této práci omezím definici na posílání dat ze serveru internetovému prohlížeči (viz. obrázek 0.1). Zdrojem dat jsou obvykle:

- video soubory (hovoříme o videu na požádání, anglicky *video on demand*),
- živé vysílání (anglicky *live streaming*).



Obrázek 0.1: Schéma streamování dat.

Publikování dat Publikování, anglicky *publishing*, označuje posílání multimediálních dat z internetového prohlížeče na server. Zdrojem dat je obvykle webkamera nebo mikrofon uživatele. Schéma publikování zachycuje obrázek 0.2.



Obrázek 0.2: Schéma publikování dat.

Kapitola 1

Analýza

Multimédia jsou obrovskou oblastí. Není možné ji v rozsahu diplomové práce celou prozkoumat, proto se tato práce zabývá relativně malou části, kterou se snaží detailně zpracovat. Věnuje se pouze problematice zvuku a videa na internetových stránkách. Technologie, které je možné použít na stránkách neurčuje server, nýbrž internetový prohlížeč. Analýza proto začíná možnostmi prohlížečů.

1.1 Technologie internetového prohlížeče

Internetové prohlížeče nabízí několik technologií pro přehrávání a publikování multimédií na HTML stránkách. Technologie se velmi odvíjí od použitého prohlížeče a nainstalovaných pluginů. Administrátoři stránek obvykle nejsou v takovém postavení, aby mohli po uživatelích vyžadovat instalaci konkrétního programu. Většinou se musí spokojit s nástroji, které má již uživatel nainstalované. Níže předkládám seznam nejběžnějších technologií pro přehrávání a publikování na internetových stránkách. Popisuji jak ty nejnovější, stejně tak ty nejstarší, které se dnes hodí spíše pro výjimečné případy.

1.1.1 HTML

Standardní HTML jazyk obsahuje několik elementů, které dokáží přehrávat video a zvuk. Žádný z nich bohužel nedokáže multimediální obsah publikovat [1]. Mohou být pro spoustu účelů dostačující, avšak nenaplňují základní požadavky této práce.

1.1.1.1 Element img

Jeden z nejstarších způsobů vložení videa do stránky je použití elementu `img`, který slouží zejména k zobrazování obrázků. Nicméně společnost Microsoft

vytvořila atribut `dynsrc`, pomocí kterého můžeme zobrazit také video [2]. Video vložíme do HTML kódu následovně:

```
<img dynsrc="source.avi" />
```

Atribut bohužel nikdy nebyl standardizován. Podporován je pouze Internet Explorerem ve verzi 2.0 a vyšší. Ostatní prohlížeče dokáží v elementu `img` zobrazovat pouze obrázky. Dnes samotný Microsoft doporučuje použít jiné způsoby pro vložení videa.

1.1.1.2 Element bgsound

Internet Explorer také přišel s nestandardním způsobem přehrávání zvuku. V HTML kódu je možné použít element `bgsound` [3]:

```
<bgsound src="source.mid" />
```

Element `bgsound` je podporován jen v Internet Exploreru, a to konkrétně od verze 2.0. V době, kdy nebylo moc možností pro vložení zvuku do stránky to byl velice populární element, nicméně dnes již existují vhodnější alternativy.

1.1.1.3 Element embed

Element `embed` nabízí způsob, jak část stránky zobrazit pluginem prohlížeče [4]. Element je určen k vložení speciálního obsahu, jako jsou Java applety, objekty ActiveX nebo Silverlight, Flash a také audio a video soubory.

```
<embed src="source.mid">

<noembed>Your browser does not support the embed tag.</noembed>
```

`Embed` je podporován všemi majoritními prohlížeči, nicméně pluginů, které dokáže spustit není mnoho. Z tohoto důvodu se element obvykle používá pouze pro vložení programů psaných v Javě nebo ve Flashi.

1.1.1.4 Element object

Nástupcem `embedu` je element `object`, který se také používá ke vkládání pluginů do internetových stránek [5]. Největší význam má atribut `classid` – říká prohlížeči jaký plugin má spustit:

```
<object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
   <param name="src" value="source.mid" />
   Your browser does not support the object tag.
</object>
```

Pokud požadované `classid` není podporováno, můžeme se pokusit obsah zpracovat jiným pluginem:

```
<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B">
    <param name="src" value="source.mov" >

    <object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
        <param name="src" value="source.mov" >

        <a href="source.mov">source.mov</a>
    </object>

</object>
```

Jestliže není podporován žádný z elementů `object`, můžeme se pokusit obsah interpretovat starším elementem `embed`. S tímto zápisem se v HTML kódu setkáme nejčastěji.

```
<object classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95">
    <param name="src" value="source.mid" />

    <embed src="source.mid" />

    <noembed>Your browser does not support neither the object tag
        nor the embed tag.</noembed>

</object>
```

Element `object` je zjednodušeně řečeno jen novější způsob zápisu elementu `embed`. Nemá vliv na nainstalované pluginy prohlížečů. Přestože některé pluginy dokáží video přehrávat i publikovat, nemáme zaručenou jejich podporu v prohlížečích. Obvykle se taktéž používá jen pro vložení Java appletů nebo Flash aplikací.

1.1.2 HTML5

Nejnovější verze jazyka HTML je označována jako HTML5. Poslední verze byla obohacena o multimediální obsah. Dokáže přehrávat zvuk i video. Dokonce nabízí rozhraní pro přístup k webkameře a mikrofonu, nicméně stále nedokáže obraz ani zvuk publikovat na server [1]. Tedy ani technologie HTML5 nesplňuje požadavky této práce.

1.1.2.1 Element video

Element `video` umožňuje přehrávat video soubor jen za pomocí internetového prohlížeče, bez vyžadování speciálního pluginu [6]. Element není vhodný pro

živé vysílání, je určen pouze k přehrávání video souborů. Video soubor může být v jednom ze 3 formátů: MP4, WebM nebo Ogg. Podpora formátů se mezi prohlížeči liší – viz. tabulka 1.1. Ukázkový příklad znázorňuje jak zobrazit alternativní obsah v prohlížečích, které element video neznají:

```
<video width="320" height="240">
    <source src="source.mp4" type="video/mp4" />
    <source src="source.webm" type="video/webm" />
    <source src="source.ogg" type="video/ogg" />

    Your browser does not support the video tag.
</video>
```

Prostřednictvím elementu `source` je možné definovat video hned v několika rozličných formátech současně. Měli bychom mít na paměti, že prohlížeč prochází zdroje v pořadí od prvního k poslednímu. Zobrazí první, který dokáže zpracovat, proto je důležité jejich pořadí. Z pravidla vkládáme zdroje podle velikosti souboru od nejmenšího k největšímu.

Tabulka 1.1: Podpora HTML5 elementu video v internetových prohlížečích.

Prohlížeč	MP4	WebM	Ogg
Internet Explorer	9+	✗	✗
Chrome	✓	✓	✓
Firefox	✓	✓	✓
Safari	✓	✗	✗
Opera	25+	✓	✓

1.1.2.2 Element audio

Element `audio` slouží k přehrávání zvukových souborů, všechny ostatní vlastnosti sdílí s elementem `video` [7]. Dostupné formáty a jejich podporu v prohlížečích zobrazuje tabulka 1.2. Zvuk se do internetových stránkách vkládá podobně jako video, význam zdrojových souborů je tedy stejný:

```
<audio>
    <source src="source.mp3" type="audio/mpeg" />
    <source src="source.wav" type="audio/wav" />
    <source src="source.ogg" type="audio/ogg" />

    Your browser does not support the audio tag.
</audio>
```

Tabulka 1.2: Podpora HTML5 elementu audio v internetových prohlížečích.

Prohlížeč	MP3	Wav	Ogg
Internet Explorer	9+	✗	✗
Chrome	✓	✓	✓
Firefox	✓	✓	✓
Safari	✓	✓	✗
Opera	✓	✓	✓

1.1.3 Pluginy prohlížečů

Téměř všechny internetové prohlížeče rozšiřují svou funkcionalitu prostřednictvím pluginů – malých spustitelných souborů, které dokáží ovládat prohlížeč i služby operačního systému. Mohou doplnit do prohlížeče libovolné funkce, včetně práce s multimédii. Největší komplikací je, že každý prohlížeč nabízí jiné rozhraní. Pokud bychom se rozhodli vytvořit vlastní plugin, bylo by nutné pro všechny nejběžnější prohlížeče udělat vlastní verzi. Poté bychom museli uživatele přesvědčit, aby si náš plugin nainstaloval. Jsou to téměř nesplnitelné úkoly, proto můžeme používat pouze pluginy, které již mají podporu v majoritních prohlížečích. Nejběžnější z nich je Silverlight, Java applet a Flash Player.

1.1.3.1 Plugin Silverlight

Microsoft Silverlight je aplikační platforma, která zdaleka nenabízí jen práci s multimediálním obsahem. Bohatě splňuje všechny požadavky práce. Původní verze pracuje pouze v operačních systémech Microsoft Windows, avšak existuje alternativa s názvem Moonlight, která je určena pro Linux. Bohužel Microsoft v květnu 2012 oficiálně ukončil práce na projektu, nemá tedy smysl pro něj vyvíjet aplikace.

1.1.3.2 Java applet

Programy v Javě se vkládají do internetových stránek pomocí tzv. *Java appletů*. Přestože existuje spousta multimediálních aplikací v Javě, pro vkládání multimédií do internetových stránek se neujala. Nejspíše je na vině samotné běhové prostředí JVM, které nenabízí potřebné knihovny. Přestože je mnoho vhodných knihoven veřejně dostupných, používají se pouze v serverových aplikacích. Jejich archivy jsou příliš velké, než aby je mohl internetový prohlížeč v rozumném čase stáhnout.

1.1.3.3 Flash Player

Flash Player je plugin internetového prohlížeče pro spouštění Flash aplikací. Funguje ve všech majoritních prohlížečích napříč operačními systémy. Nabízí

nástroje pro pohodlné streamování a publikování multimédií. V tuto chvíli je to jediné vhodné řešení, proto bude software vytvořený pro tuto práci stavět právě na technologii Flash. Podrobněji se Flashi a souvisejícím technologiím věnuji v následujících kapitolách.

1.2 Flash

Flash je softwarová platforma pro tvorbu grafiky a aplikací. Prosadil se jako alternativa animovaným obrázkům GIF. Dnes se používá obzvláště pro vektorové a rastrové animace, interaktivní prezentace, online hry a webové aplikace. Autorem je společnost Macromedia, avšak od roku 2005 je vyvíjen společností Adobe.

Macromedia vytvořila pro Flash vlastní skriptovací jazyk ActionScript, jehož syntaxe se podobá JavaScriptu [8]. Do dnešní doby byly publikovány 3 verze. Nejnovější nese označení ActionScript 3.0. Původní vývojové prostředí se jmenovalo Macromedia Flash, novější verze jsou vyvíjeny pod názvem Adobe Flash Professional.

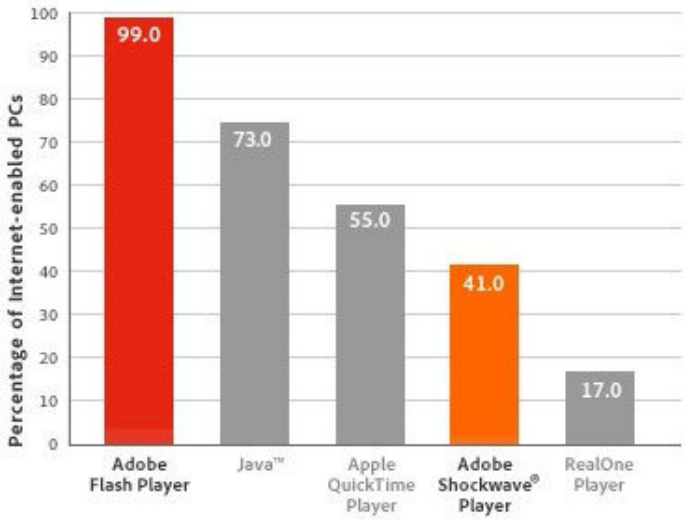
Aplikace se distribuují v binárních souborech, které jsou obvykle velmi malé. Například běžný přehrávač videa má řádově jen několik desítek, maximálně stovek kB. Soubory interpretuje Flash Player. Nejčastěji se s ním setkáme v podobě pluginu pro internetový prohlížeč, který spouští aplikace přímo uvnitř stránek. Není to ovšem podmínkou, skripty je možné také publikovat jako desktopové aplikace nebo aplikace pro mobilní zařízení.

Flash používá pro přehrávání videa např. služba YouTube nebo v českém prostředí Stream. Oficiální materiály Adobe uvádí dostupnost Flashe v 99% počítačů připojených k internetu (viz. graf 1.1). V případě mobilních zařízení je to téměř 70% (viz. graf 1.2). Odhady společnosti Adobe budou možná příliš optimistické, nicméně v roce 2004 server NaVrcholu.cz změřil podporu Flash doplňku v prohlížečích okolo 90% [9]. Novější důvěryhodný zdroj dat se mi nepodařilo dohledat, avšak s největší pravděpodobností se od té doby plugin ještě více rozšířil.

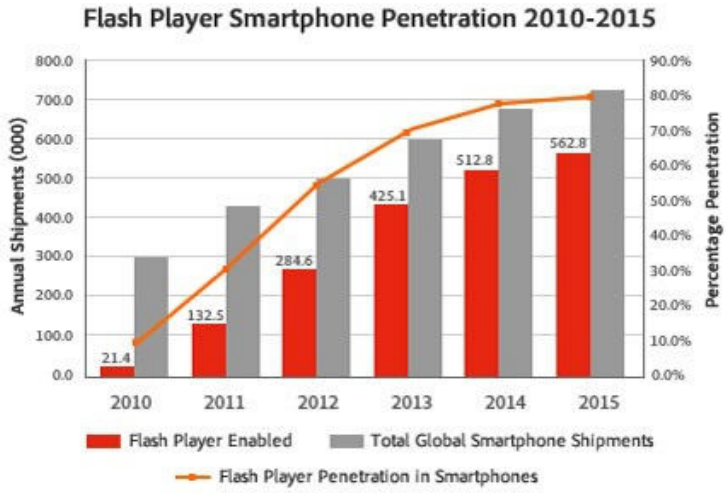
Flash je možné používat k rozmanitým činnostem. Pro tuto práci jsou důležité zejména schopnosti při spolupráci se serverem:

- přehrávání obrazových a zvukových dat ze serveru jako živé vysílání i video na požádání (VOD),
- publikování obrazu z webkamery a zvuku z mikrofonu uživatele na server.

Pro komunikaci se serverem používá Flash protokol RTMP.



Obrázek 1.1: Podpora technologie Flash v počítačích připojených k internetu.
Zdroj: https://www.adobe.com/cz/products/flashplatformruntimes/statistics.html



Obrázek 1.2: Podpora technologie Flash v mobilních zařízeních.
Zdroj: https://www.adobe.com/cz/products/flashplatformruntimes/statistics.html

1.3 RTMP protokol

Real Time Messaging Protocol, zkráceně RTMP, je binární síťový protokol z dílny společnosti Adobe [10]. Využívá vrstvu TCP/IP na portu 1935. Protokol dokáže přenášet zvuková, obrazová a strukturovaná data. Slouží k obousměrné komunikaci mezi Flash Playerem a oficiálním serverem *Adobe Media Server*, zkráceně AMS [11]. Protokol byl původně chráněn standardem, po jeho uvolnění a zveřejnění specifikace vzniklo mnoho alternativních open source implementací. S protokolem RTMP úzce souvisí několik odnoží:

RTMPS – Verze RTMP využívající šifrování pomocí TLS/SSL vrstvy.

RTMPE – Verze RTMP využívající šifrovací mechanismus vyvinutý společností Adobe, který je předmětem průmyslového standardu.

RTMPT – Tunelování základní verze protokolu přes HTTP požadavky. Standardně využívá port 80, příp. 443 pro tunelování RTMPS.

RTMFP – UDP obdoba protokolu RTMP.

Tato práce se však zabývá pouze RTMP. Nejvíce se cílům práce blíží RTMPS, který pouze přidává podporu pro šifrování. Ostatní protokoly se v praxi příliš nepoužívají.

Protokol RTMP je rozdělen do několika vrstev:

binární data – Na nejnižším stupni jsou data v binárním tvaru posílaná protokolem TCP.

vrstva chunků – Binární data se spojují do `chunků`, bloků dat pevné délky. Několik `chunků` dohromady vytvoří jeden `tag` – samostatný `chunk` nemá význam. `Chunky` různých `tagů` se mohou libovolně kombinovat, takže v kanálu mohou být části několika `tagů` současně.

vrstva tagů – Základní logická jednotka protokolu je `tag`. Obsahuje informace o svém typu (`type`), času (`timestamp`) a logickém streamu (`streamID`).

vrstva message – Struktura obsahující všechny dostupné údaje obsažené v protokolu se nazývá `message`. Většina tříd pracuje právě se strukturou `message`, která se dělí na podtypy:

- `SetChunkSize`
- `Abort`
- `Acknowledgement`
- `Control`

- `SetWindowAcknowledgementSize`
- `SetPeerBandwidth`
- `Audio`
- `Video`
- `Meta`
- `SharedObject`
- `Command`

Každý podtyp obsahuje jiné vlastnosti, např. pro `SetChunkSize` jsou to: `timestamp`, `streamID` a `chunkSize`.

streamy – Každá `message` patří do nějakého logického streamu. V jednom spojení může v jednu chvíli existovat několik streamů, takže uživatel může souběžně přehrávat několik videí a ještě přitom nahrávat vlastní záznam. Data mohou být posílaná oběma směry – jak od klienta serveru, tak od serveru ke klientovi.

Za zmínku stojí, že audio a video data jsou komprimovaná kodeky. Není možné s nimi manipulovat, ve zprávách jsou reprezentována pouze jako sekvence bytů. Pokud bychom chtěli pracovat se zvukem nebo obrazem, je nutné je nejprve dekódovat. Ve většině případů to však není potřeba. Streamovací server v principu jen posílá data ze souborů klientovi, ukládá data od klienta do souboru nebo data přeposílá mezi klienty navzájem, aniž by chápal jejich obsah.

1.4 Formát AMF

Action Message Format, zkráceně AMF, je binární formát pro serializaci dat [12][13]. Používá se zejména v prostředí technologie Flash. Setkat se s ním můžeme nejčastěji při komunikaci po síti protokolem RTMP, ale také ve FLV souborech pro ukládání metadat. Je to také standardní formát pro serializaci objektů ActionScriptu.

AMF prezentuje současně s daty také jejich datový typ. Serializovat dokáže všechny datové typy ActionScriptu a to jak ve stromové struktuře, tak díky referencím také ve struktuře grafu.

Společnost Adobe publikovala AMF v roce 2001 ve Flash Playeru 6. S příchodem Flash Playeru 9 a ActionScriptu 3.0 vydala novou verzi AMF3. Původní verzi přejmenovala na AMF0.

1.4.1 Formát AMF0

Seznam podporovaných typů původního formátu obsahuje tabulka 1.3. Setkat se s ním můžeme v metadatech FLV souborů a v RTMP protokolu ve zprávách:

- `Meta` (`0x12`),
- `Command` (`0x11`),
- `SharedObject` (`0x13`).

Datový typ	Popis
Double	Číslo v plovoucí řádové čárce s dvojitou přesností.
Boolean	Logická hodnota true nebo false.
String	Řetězec v UTF8.
Object	Asociativní pole.
MovieClip	Vyhrazeno pro budoucí použití.
NULL	Prázdná hodnota.
Undefined	Nedefinovaná hodnota (validní ActionScript typ).
Reference	Reference na jiný serializovaný objekt (umožňuje zmenšit velikost dat a reprezentovat grafy).
EcmaArray	Asociativní pole (včetně velikosti pole).
Date	Datum a čas s přesností na milisekundy.
UnsupportedType	Neserializovatelný objekt.
RecordSet	Vyhrazeno pro budoucí použití.
XML	XML dokument reprezentovaný řetězcem v UTF8.
TypedObject	Objekt včetně jeho názvu třídy.

Tabulka 1.3: Datové typy formátu AMF0.

1.4.2 Formát AMF3

AMF3 je výchozí formát pro ActionScript 3.0 [13]. Soubory FLV sice není podporován, zato je běžnější v RTMP protokolu. Používá se ve 3 typech zpráv:

- `Meta` (`0x0F`),
- `Command` (`0x14`),
- `SharedObject` (`0x10`).

Formát přináší nové typy, které v původní verzi nebyly dostupné. Oproti AMF0 je úspornější, dokáže data reprezentovat za použití stejného nebo menšího množství bytů. Čísla používají formát, který mění svou velikost v závislosti na hodnotě, obdobně jako např. v UTF8 znaky. Změny oproti původní verzi jsou k vidění v tabulce 1.4.

Tabulka 1.4: Změny ve formátu AMF3 oproti AMF0.

Datový typ	Popis
Integer	Nový typ pro celá čísla (používá proměnný počet bytů); Double je stále zachován pro čísla v plovoucí řádové čárce (používá vždy 8 bytů).
Boolean	Logické hodnoty kódovány pouze jedním bytem, oproti původním dvěma.
String	Velikost řetězce je reprezentována s proměnnou délkou.
Object	Přidává k objektu název třídy.
MovieClip	Odstraněn.
Reference	Nově může být reference také na řetězec.
EcmaArray	Funkce asociativního pole nezměněna, pouze přejmenován na Array.
UnsupportedType	Odstraněn.
RecordSet	Odstraněn.
ByteArray	Nový typ pro binární data.
TypedObject	Odstraněn (funkce nahrazena vylepšeným typem Object).

1.4.3 Kombinování AMF0 a AMF3

RTMP server musí podporovat oba formáty. Není možné se rozhodnout např. pouze pro AMF3 a první verzi ignorovat. Formáty se totiž vzájemně prolínají:

- AMF3 definuje příkaz s kódem `0x00`, který „přepne“ do formátu AMF0. Za tímto kódem následuje sekvence bytů ve formátu AMF0.
- AMF0 byl zpětně doplněn o typ `AMV Plus` s kódem `0x17`, za nímž následují data formátu AMF3.

1.5 Node.js

Dle zadání práce budu formát AMF a protokol RTMP implementovat v Node.js. Jedná se o open source platformu pro tvorbu aplikací v JavaScriptu [14]. Node.js je založen na událostmi řízeném programování, které je vhodné pro datově náročné realtime aplikace. Využívá interpret V8 Engine vytvořený společností Google, který je součástí také jejich internetového prohlížeče Google Chrome [15]. Domovská stránka projektu je `http://nodejs.org/`.

Nejčastěji se Node.js používá pro tvorbu serverových aplikací. Není to však podmínkou, stejně dobře může posloužit v klientském nebo jiném prostředí. Samotný V8 Engine lze snadno implementovat do vlastních aplikací a rozšířit

je tak o ovládání prostřednictvím interpretovaného JavaScriptu. Podporovány jsou operační systémy: OS X, Microsoft Windows, Linux a FreeBSD.

Node.js nabízí mj. rozhraní pro práci s procesy, souborovým systémem, sítí, pamětí a časovači [16]. Vlákna nejsou podporována. Celý program pracuje jako jedno vlákno, které obsluhuje smyčku událostí, tzv. *Event Loop*. Dlouhotrvající činnosti jsou zpracovávány asynchronně, provádějí je vlákna ovládaná V8 Enginem na pozadí. Všechny události (jako je například otevření souboru, otevření nového socketu, dokončení zápisu dat do souboru, uplynutí požadované doby časovače aj.) jsou vkládány do smyčky událostí, ze které jsou zpracovány jedna po druhé hlavním vláknem.

1.6 Node.js moduly

Aplikace v Node.js členíme do tzv. *modulů* – samostatných funkčních celků. Moduly jsou psány obvykle v JavaScriptu. V některých specifických případech musí být modul napsán v C++ a před spuštěním zkompilován – například pokud potřebujeme připojit do modulu externí knihovnu, využít služby operačního systému nebo optimalizovat výkon. Pro označení kompilovaných modulů je běžnější anglický termín *addon* namísto běžného *module* [17]. V této práci si dovolím v obou případech používat označení modul. V českém prostředí se dosud nevžil vhodný překlad.

V oficiálním repozitáři `http://npmjs.org/` je mnoho volně dostupných modulů. Kolem Node.js působí velká vývojářská komunita. Vzniklo a stále vzniká obrovské množství projektů. Pokud narazíme při tvorbě internetové stránky na problém, s největší pravděpodobností na to již existuje řešení v repozitáři. Bohužel většina modulů se nikdy nedostane do stabilní verze. Zcela běžné jsou moduly ve verzi 0.0.1 apod.

Zdrojový kód této práce budu členit do modulů. Při jejich návrhu kladu důraz na univerzálnost tak, aby mohl každý z nich v maximální míře pracovat samostatně. Celkem tedy vytvořím 3 moduly:

RTMP modul – Implementace protokolu RTMP pro tvorbu serveru, streamování a publikování dat, zápis a čtení video souborů.

AMF modul – Implementace AMF0 a AMF3 pro serializaci a deserializaci dat. Modul bude na ostatních zcela nezávislý, bude jej možné využít také v jiných projektech.

AudioVideo modul – Práce s multimediálními kodeky, kódování a dekódování zvukových a obrazových dat.

1.7 RTMP a AMF modul

V repozitáři NPM existuje několik modulu zabývajících se AMF a RTMP. V tabulce 1.5 předkládám souhrnné informace o těch, které jsem podrobil analýze. Bohužel, jak značí jejich verze, většina z nich je nestabilní, autory nedoporučovaná k užití. Podporují pouze základy RTMP. Žádný z modulů neimplementujte kompletní specifikaci AMF0 ani AMF3. Zdrojové kódy jsou až na výjimky netestované. Jako nejvhodnější modul pro splnění vytyčených cílů jsem vybral node-rtmpapi – domovská stránka projektu je `https://www.npmjs.com/package/node-rtmpapi`. Modul poslouží jako základ práce, zvolil jsem jej pro jeho přednosti:

- podpora RTMP, AMF0 i AMF3,
- snadná rozšířitelnost,
- srozumitelný zdrojový kód.

Naopak mezi jeho hlavní nedostatky patří:

- nepodporované RTMP zprávy `Audio`, `Video`, `SharedObject` a `Aggregate`,
- nedokáže dekódovat všechny příkazy AMF0 a AMF3,
- nestandardní konvence pojmenování funkcí.

Pro správnou implementaci bude nutné tyto nedostatky odstranit a zdrojový kód otestovat.

Název	Verze	Jazyk	RTMP	AMF0	AMF3
node-rtmpapi (využívá node-amfutils)	0.0.1	JavaScript	částečně	✗	✗
node-amf	1.0.1	JavaScript	částečně	částečně	částečně
node-amfutils	0.0.1	JavaScript	✗	částečně	částečně
node_amf_cc	1.1.1	C++	✗	✗	částečně
amf-struct	0.1.0	CoffeeScript	✗	částečně	částečně
amf_deserializer	0.0.6	JavaScript	✗	částečně	✗
amf	0.1.0	JavaScript	✗	částečně	částečně

Tabulka 1.5: Přehled Node.js modulů podporujících RTMP, AMF0 a AMF3.

1.8 AudioVideo modul

V repozitáři existuje také několik modulů pracujících s obrazovými a zvukovými daty. Provedl jsem jejich analýzu, ze které vyplývá, že se dělí na 2 skupiny:

Specializované – Moduly podporující jeden zvukový nebo obrazový formát.

Univerzální – Moduly podporující několik desítek formátů. Pro svou funkci využívají nějakou komplexní knihovnu.

Flash podporuje mnoho zvukových a obrazových kodeků. Není možné každý z nich implementovat pomocí vlastního specializovaného modulu. Pro tuto práci mají smysl pouze univerzální moduly.

Blíže jsem prostudoval univerzální moduly: navcodec, Fluent-ffmpeg, ffmpeg-binary-linux-64bit, ffmetadata, ffmpeg a codem-transcode. Všechny až na jeden pouze obalují nástroj FFmpeg spouštěný z příkazové řádky, výjimkou je modul navcodec, který provádí totéž s nástrojem Libav. Moduly pracují na principu dávkového zpracování:

1. nastaví vhodné argumenty,
2. spustí příkaz,
3. zpracují výstup.

Tyto modely z principu nemohou interaktivně upravovat data, nesplňují tedy požadavky této práce. Z analýzy vyplývá, že je nutné vytvořit nový modul, do kterého bude přímo začleněna vhodná grafická knihovna tak, aby mohly být její funkce volány interaktivně. Jako nejlepší knihovnu jsem vybral právě Libav, kterou používá jeden ze zkoumaných modulů. Bohužel aby bylo možné knihovnu s modulem propojit, musí být modul naprogramován v C++ namísto JavaScriptu.

1.9 Knihovna Libav

Libav je multiplatformní knihovna určena pro práci s audiovizuálními daty [18]. Podporuje několik desítek video kodeků, zvukových kodeků a multimediálních kontejnerů. Její zdrojový kód je v jazyce C, avšak některé části jsou optimalizovány v jazyce symbolických adres (někdy nepřesně označovaném assembler). Knihovna je známá spíše nepřímo a to díky nástroji FFmpeg, ze kterého se v roce 2009 odtrhla část vývojářů, aby založila projekt Libav [19]. Stejně jako FFmpeg i Libav tvoří několik nástrojů spustitelných z příkazové řádky [18]:

avconv – Vychází z `ffmpeg`; provádí konverze formátů.

avplay – Vychází z `ffplay`; desktopový přehrávač.

avprobe – Vychází z `ffprobe`; slouží k extrakci informací z multimédií.

avserver – Vychází z `ffserver`; obsahuje nástroje pro tvorbu serveru.

Libav není jeden monolitický celek. Skládá se z několika menších knihoven:

Libswscale – Vysoce optimalizované konverze obrazu, zejména změna rozlišení (zvětšení nebo zmenšení rozměrů), změna barevné hloubky a formátu pixelů (například z `yuv420p` na `rgb24`).

Libswresample – Vysoce optimalizované převzorkování zvuku (například změna vzorkovací frekvence z 44100Hz na 8000Hz), změna uspořádání kanálů (například ze stereo na mono) a další konverzní operace.

Libavutil – Užitečné funkce s jednotným chováním napříč operačními systémy. Obsahuje nástroje pro bezpečnou práci s řetězci, datovými strukturami, matematickými funkcemi, kryptografií, generátory náhodných čísel a funkce související s multimédii. Knihovna se skládá z několika nezávislých částí, které je možné přidat nebo odebrat při kompilaci.

Libavfilter – Zvukové a obrazové filtry.

Libavformat – Multiplexování a demultiplexování dat.

Libavcodec – Podpora kodeků a multimediálních kontejnerů. Slouží ke kódování a dekódování.

Libavdevice – Přístup k vstupně/výstupním zařízením nezávisle na operačním systému nebo hardwarové architektuře.

Libav je distribuována v podobě zdrojových kódů nebo již zkompilovaných dynamických knihoven. Je možné ji poměrně snadno vložit do vlastních programů. Tato knihovna splňuje kladené požadavky na AudioVideo modul [20]. S její pomocí dokáže kódovat a dekódovat audiovizuální data, která jsou pro modul RTMP pouze nesrozumitelnou sekvencí bytů.

1.10 Shrnutí analýzy

Z analýzy klientských technologií vyplynulo, že standardní HTML elementy prohlížeče jsou vhodné pouze pro přehrávání videa. Publikovat data z webkamery a mikrofonu uživatele na server dokáží pouze některé pluginy. V dnešní době je tím nejvhodnějším Flash Player. Dokáže splnit cíle práce a mimoto je podporován drtivou většinou prohlížečů.

Vyvíjet budu pro serverovou platformu Node.js, která strukturuje zdrojový kód do modulů. První modul, který je nutné vytvořit, se jmenuje RTMP. Protokolem RTMP komunikuje Flash aplikace uvnitř internetové stránky se serverem. Tento modul bude implementovat všechny funkce protokolu. Druhý modul bude sloužit k serializování strukturovaných dat do AMF formátu, který je součástí RTMP. S tvorbou modulů nebudu začínat na zelené louce. Zpracoval jsem rešerši existujících řešení. Nejvíce se požadavkům blíží moduly node-rtmpapi a node-amfutils, jež se stanou základem pro tvorbu vlastních modulů.

RTMP video ani zvuk neposílá v surové podobě. Data by byla příliš objemná, proto jsou komprimována. Rekonstruovat je do původní podoby vyžaduje podporu kodeků. Protokol RTMP podporuje celkem 9 kodeků videa a 10 kodeků zvuku. Není v rozsahu této práce je všechny implementovat, nebylo by to ani rozumné. Třetí modul s názvem AudioVideo bude podporu kodeků zajišťovat pomocí knihovny Libav. Na rozdíl od prvních dvou modulů, které budou programovány v JavaScriptu, musí být třetí modul vytvořen v C++, což přináší komplikace při vývoji i následné instalaci.

Kapitola 2

Návrh

Na základě analýzy jsem navrhl softwarovou architekturu rozdělenou do 3 modulů:

1. AMF,
2. RTMP,
3. AudioVideo.

Ke každému modulu předkládám diagram nejdůležitějších tříd a stručně popisuji jeho funkci. Doprovodné obrázky zachycují základní schéma komunikace mezi třídami. Aby byly obrázky srozumitelné, musel jsem schémata zjednodušit. Podrobnější informace jsou dostupné v dokumentaci.

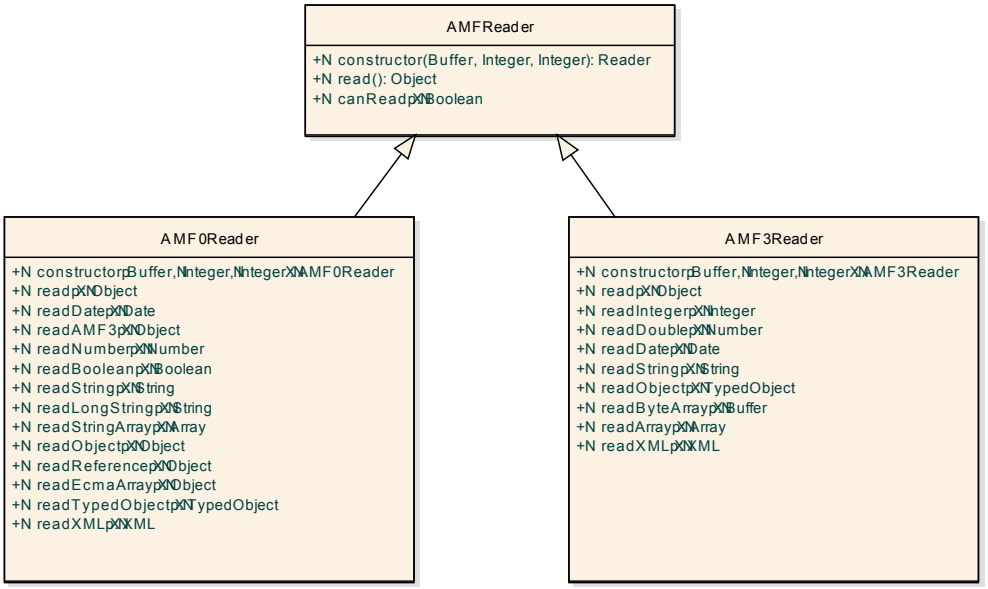
Pro upřesnění: JavaScript nepoužívá dědění založené na třídách, nýbrž prototypovou dědičnost. V této práci si přesto dovolím používat termín třída, protože předpokládám, že bude pro čtenáře srozumitelnější. Současně nemůže vzniknout žádné nedorozumění.

2.1 Návrh AMF modulu

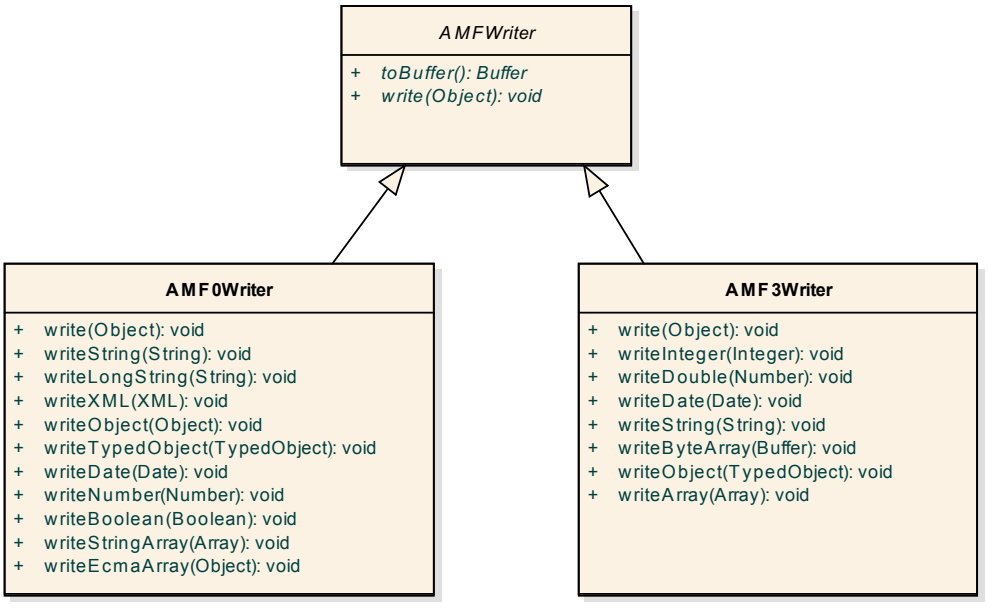
Účelem modulu je serializování a deserializování AMF hodnot. Díky tomu, že se ActionScript velice podobá JavaScriptu, je možné většinu datových typů jednoduše namapovat na sebe. Pouze pro typy, ve kterých se liší, jsem vytvořil vlastní třídy. Celkově se modul skládá ze tříd 3 typů:

AMF*Reader – Deserializace dat z bytů. Diagram tříd AMFReader, AMF0Reader a AMF3Reader zachycuje obrázek 2.1. Jejich struktura je jednoduchá, zajímavé jsou spíše jejich metody.

AMF*Writer – Serializace dat do bytů, tedy inverzní operace k deserializování. Rozhraní tříd AMFWriter, AMF0Writer a AMF3Writer zobrazuje diagram 2.2.



Obrázek 2.1: Diagram tříd AMF modulu pro serializaci.



Obrázek 2.2: Diagram tříd AMF modulu pro deserializaci.

Ostatní – Některé datové typy jsou podporované ActionScriptem, avšak nemají ekvivalent v JavaScriptu. Aby je bylo možné reprezentovat, vytvořil jsem pro ně třídy: RecordSet, TypedObject, UnsupportedType a XML. Nepřikládám diagram tříd, protože jejich struktura je příliš jednoduchá. TypedObject a XML definují pouze settery a gettery. Ostatní třídy nemají žádné metody.

AMF je zcela nezávislý na ostatních modulech, takže je možné jej použít i v jiných projektech.

2.2 Návrh RTMP modulu

Modul implementuje RTMP protokol a všechny jeho funkce nabízí prostřednictvím intuitivního rozhraní. Jeho nejdůležitějším úkolem je tvorba streamovacího serveru a obsluha síťových klientů. Některé části protokolu obsahují AMF data, proto je modul RTMP závislý na modulu AMF.

2.3 Návrh filtrů RTMP modulu

Protokol RTMP se skládá z vrstev. Přirozeně jsem rozdělil také modul do vrstev. Navrhl jsem několik na sobě nezávislých jednoúčelových filtrů, každý z nich obsluhuje jednu vrstvu. Filtry mohou být libovolně přidávány nebo odebírány tak, aby byla výstupní data v požadovaném tvaru. Filtry si předávají data pomocí struktur:

Buffer – Binární data.

Tag – Reprezentuje RTMP strukturu `tag`.

Message – Reprezentuje RTMP strukturu `message`.

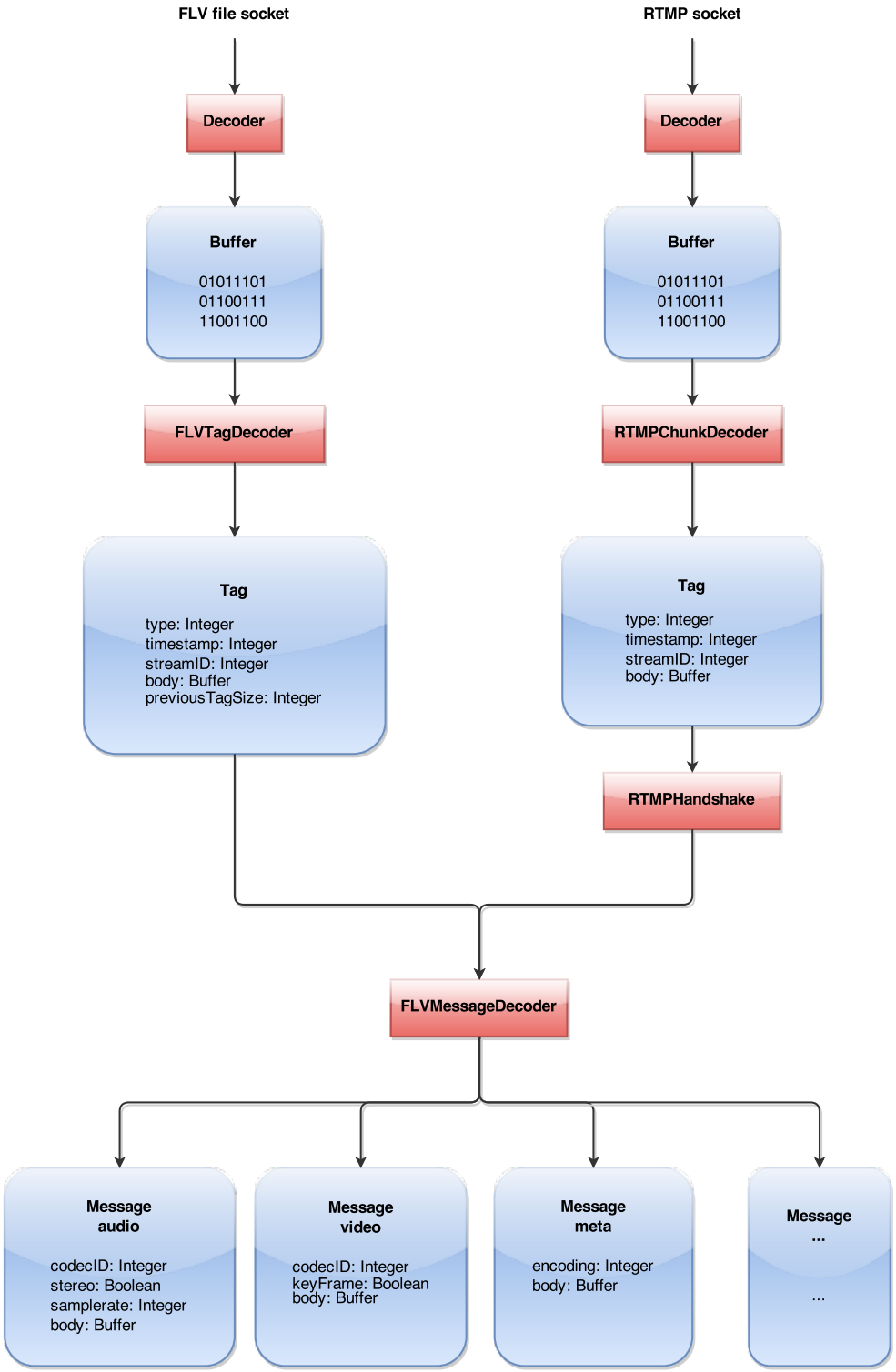
Obrázek 2.3 zobrazuje vlastnosti, které tyto struktury obsahují a způsob, jakým jsou předávány mezi filtry. Při návrhu struktur i filtrů jsem vycházel z činnosti protokolu.

2.3.1 Dekódovací filtry

Transformace binárních dat na strukturu `Message` provádí dekódovací filtry:

Decoder – Základní filtr, který nabízí rozhraní pro asynchronní čtení binárních dat ze standardního Streamu. Musí být použit pro správnou funkci ostatních filtrů.

RTMPChunkDecoder – Transformuje binární data v RTMP na `Tagy`. Používá se pro dekódování RTMP protokolu.



Obrázek 2.3: Schéma dekódovacích filtrů RTMP modulu.

FLVTagDecoder – Transformuje binární data ve FLV na `Tagy`. Používá se pro dekódování FLV souborů.

FLVMessageDecoder – Transformuje `Tagy` na `Message`, doplňuje je tedy o další informace závislé na typu.

AMFMessageDecoder – Dekóduje AMF data ve zprávách `Meta`, `Command` a `SharedObject`.

Obrázek 2.3 zachycuje proces dekódování. Opačný postup, tedy kódování, provádí druhá skupina filtrů.

2.3.2 Kódovací filtry

Kódování `Message` na binární data provádí filtry:

Encoder – Základní filtr, který nabízí rozhraní pro asynchronní zápis binárních dat do standardního streamu. Musí být použit pro správnou funkci ostatních filtrů.

RTMPTagEncoder – Transformuje `Tag` na sekvenci bytů RTMP protokolu.

FLVTagEncoder – Transformuje `Tag` na sekvenci bytů FLV souboru.

FLVMessageEncoder – Transformuje `Message` na `Tag`.

AMFMessageEncoder – Kóduje AMF typy: `Meta`, `Command` a `SharedObject`.

2.3.3 Filtry pro navázání spojení

Dva speciální filtry pro navázání spojení, tzv. *handshake*:

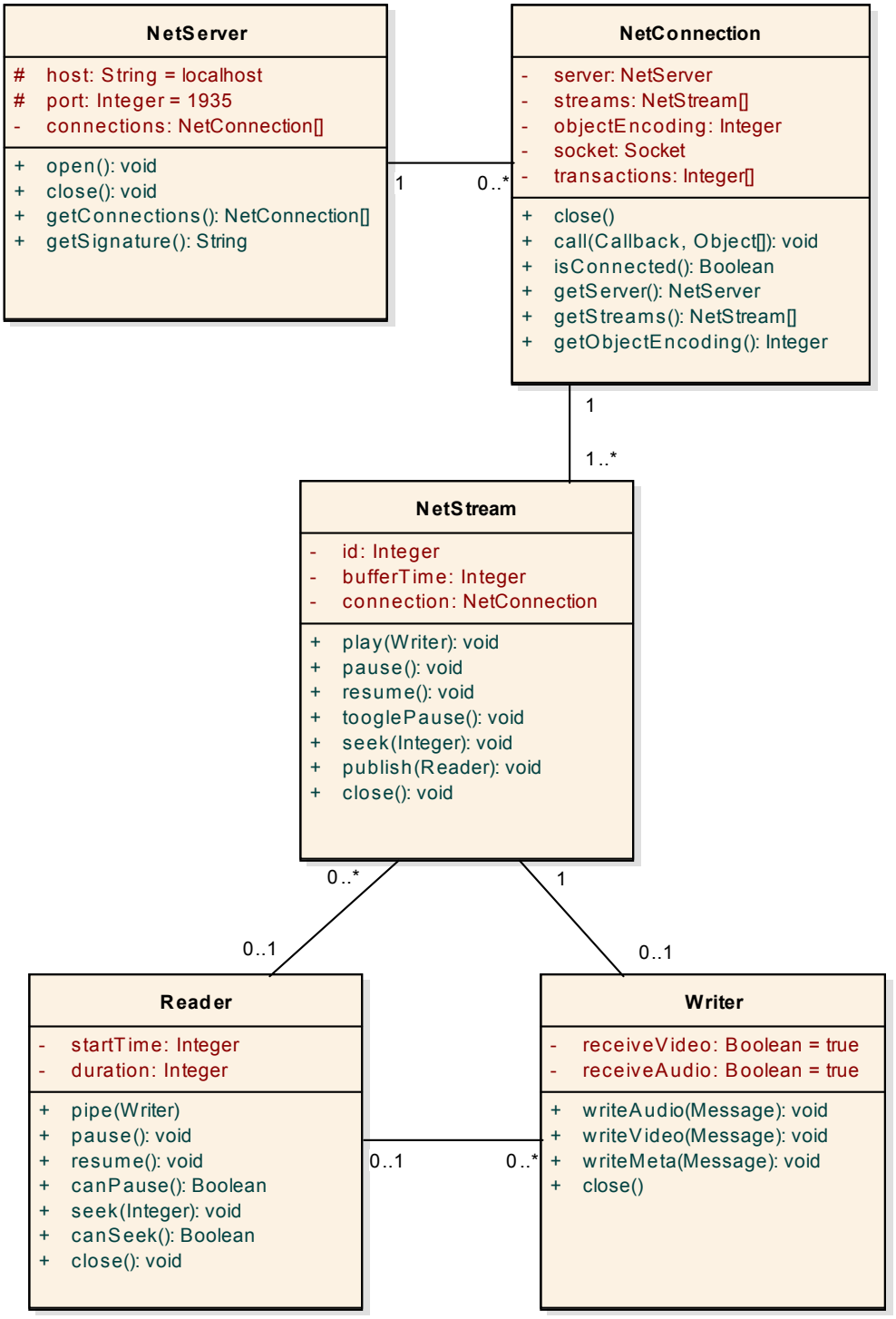
RTMPClientHandshake – Navázání spojení s RTMP serverem.

RTMPServerHandshake – Navázání spojení s RTMP klientem.

2.4 Návrh tříd RTMP modulu

Filtry jsou vhodným nástrojem pro jednoduché činnosti, pro ty komplikované se naopak nehodí. Pro komplexní úkoly jsem navrhl třídy. Mezi ty nejdůležitější patří:

NetServer – RTMP server. Naslouchá na síťovém rozhraní na nové klienty. Není-li uvedeno jinak, standardně používá port 1935 a IP adresu `127.0.0.1`. K serveru se zpravidla připojuje Flash Player z internetového prohlížeče. Při navázání nového spojení vygeneruje server událost `(on)connection`, která slouží k obsluze klienta.



Obrázek 2.4: Diagram tříd RTMP modulu.

NetConnection – Spojení mezi klientem a serverem prostřednictvím protokolu RTMP. Nabízí rozhraní pro přenos multimediálních dat prostřednictvím NetStreamů, vzdálené volání procedur (RMI) a správu sdílených objektů. NetConnection je abstraktní třída, kterou rozšiřuje ServerNetConnection a ClientNetConnection.

ClientNetConnection – Spojení používané pokud Node.js vystupuje jako RTMP klient, který se připojuje k serveru. Obdoba NetConnection v ActionScriptu. Nabízí metody `publish` (pro publikování videí na server) a `play` (pro přehrávání videí ze serveru).

ServerNetConnection – Spojení používané pokud Node.js vystupuje jako RTMP server. Server nemůže sám od sebe začít přenášet data, všechny akce musí iniciovat klient, proto neobsahuje ani metodu `publish` ani `play`. Namísto toho generuje události `(on)publish`, pokud klient chce publikovat data z webkamery nebo mikrofonu, a `(on)play`, pokud klient žádá o přehrání videa nebo připojení k živému streamu.

NetStream – Multimediální stream. Obdoba NetStream v ActionScriptu. Přenáší jeden multimediální proud dat, např. video nebo živé vysílání. Na stream mohou být aplikovány filtry, které upravují jeho zvuková a obrazová data.

Reader – Třídy generující data pro NetStream. Tento projekt vyžaduje pouze FileReader, který čte data ze souboru. V jiných situacích může najít uplatnění např. CameraReader (čtení z webkamery), RTMPReader (čtení z jiného RTMP spojení), ListReader (spojení několika Readerů do playlistu) a další.

Writer – Třídy pro ukládání příchozích dat z NetStreamu. Tento projekt vyžaduje pouze FileWriter, který zapisuje data ze souboru. V jiných situacích může najít uplatnění např. RTMPWriter (streamování do jiného RTMP spojení), HTTPLSWriter (streamování prostřednictvím HTTP) a další.

Provázanost základních tříd zachycuje diagram 2.4. Třídy využívají kódovacích a dekódovacích filtrů. Každá třída, podle své funkce, volí vhodné filtry a aplikuje je na stream. Nicméně na streamy mohou být aplikovány taky další filtry, které mohou upravovat data nebo filtry AudioVideo modulu.

2.5 Návrh video souborů

RTMP server musí umět pracovat se soubory. Ke čtení `tagů` ze souborů slouží třída FileReader a k zápisu do souborů FileWriter. Nenavrhoval jsem žádný speciální formát souborů, používám standardní kontejner Flash Video, protože je velice podobný RTMP protokolu [21]. Soubory se ukládají s příponou `.flv`. Skládají se z FLV hlavičky a FLV `tagů`.

2.5.1 Hlavička FLV souborů

Každý FLV soubor musí začínat hlavičkou dlouhou 9 bytů [21]. Její strukturu ukazuje tabulka 2.1. Význam jednotlivých bytů:

1. – 3. byte – V prvních třech bytech je uložena reprezentace ascii znaků „FLV“. Mnoho kontejnerů začíná soubor krátkým identifikátorem, aby např. video přehrávače dokázaly spustit i soubor bez přípony.

4. byte – Další byte obsahuje použitou verzi. Podporována je pouze verze 1 s hodnotou `0x01`.

5. byte – Následuje byte příznaků. Video reprezentuje nejnižší bit, audio 3. nejnižší bit. Audio soubory mají tedy příznak `0x04`, video soubory `0x01` a soubory obsahující audio i video jejich bitový součet `0x05`.

6. – 9. byte – Poslední čtyři byty obsahují velikost hlavičky reprezentované 32bitovým celým číslem bez znaménka v kódování big endian. Prostor 4 bytů je zcela zbytečně předimenzovaný, hlavička je vždy dlouhá pouze 9 bytů.

RTMP modul vytváří soubory pouze s příznakem `0x05`, tedy kombinace audia a videa, a to i v případě, když obsahují jen zvuk nebo jen obraz. Přestože neporušuje specifikaci, některým programům to způsobuje problémy. Např. VLC přehrávač při otevření souboru analyzuje několik prvních `tagů`, aby identifikoval zvukový a obrazový kodek. Jestliže se mu podaří najít jen z nich, zobrazí uživateli chybovou hlášku, nicméně soubor se normálně přehraje.

1.	2.	3.	4.	5.	6.	7.	8.	9.
`"F"`	`"L"`	`"V"`	`0x01`	`0x05`	`0x00`	`0x00`	`0x00`	`0x09`

Tabulka 2.1: Hlavička FLV souborů.

2.5.1.1 Obsah FLV souborů

Za hlavičkou následují FLV `tagy`. Jejich formát je stejný jako u RTMP `tagů`, nicméně FLV soubor může obsahovat pouze malou podmnožinu typů. Podporovány jsou pouze `tagy`:

- `Meta`,
- `Audio`,
- `Video`.

`Tagy` se mohou v souboru vyskytovat zcela libovolně. Není zaručeno jejich pořadí, avšak obvykle je první `tag` `Meta`, obsahující metadata, pokud je video obsahuje. `Meta` se používá také uvnitř souboru pro vkládání strukturovaných dat – např. titulky nebo řídící příkazy.

2.5.1.2 Metadata video souborů

Formát metadat není pevně určen, nicméně měl obsahovat alespoň následující informace [22]:

videocodecid – Identifikátor video kodeku, kterým jsou kódována obrazová data. Flash Player dokáže přehrát i video, které nemá uvedený kodek nebo je uveden špatně. Nicméně některé desktopové přehrávače video soubor bez uvedeného kodeku nepřehrají.

audiocodecid – Identifikátor audio kodeku, kterým jsou kódována zvuková data. Platí pro ně totéž, jako pro obrazová data. Flash Player dokáže přehrát i zvuk, který nemá uvedený kodek nebo je uveden špatně. Nicméně některé desktopové přehrávače zvukový soubor bez uvedeného kodeku nepřehrají.

duration – Délka souboru v sekundách.

Dále metadata běžně obsahují informace o obraze (`width`, `height`, `framerate` a `videodatarate`) a zvuku (`stereo`, `audiosamplerate`, `audiosamplesize` a `audiodatarate`), nicméně Flash Player je k ničemu nevyužívá. Ostatní informace jsou zcela volitelné, můžeme si dokonce definovat své vlastní.

Při přehrávání se soubory čtou sekvenčně – od začátku do konce. Nicméně za standard se dnes považuje schopnost přehrávače „skočit“ na jinou pozici, tzv. *seekování*. Uživatel kliknutím na časovou osu vyžádá po serveru přesunutí na tuto pozici. Aby mohl přehrávač zobrazit časovou osu, musí znát délku videa a proto je vlastnost `duration` v metadatech nezbytná. Ve všech souborech RTMP modulu musí být délka videa v metadatech uvedena.

S tím souvisí problém: při ukládání videa publikovaného uživatelem, server neví jak bude dlouhé. Neví to ani sám uživatel. Video skončí, až přestane

nahrávat. Teprve v tu chvíli se server dozví, jak je video dlouhé a můžu tuto informaci zapsat do hlavičky. Operační systém neumožňuje zapisovat data na začátek souboru tak, aby se všechna ostatní posunula. Je nutné provést komplikovaný a výpočetně náročný proces:

1. vytvořit nový soubor,
2. zapsat do něj metadata,
3. nakopírovat všechny audio a video `tagy` z původního souboru (který může mít běžně několik desítek nebo stovek MB),
4. původní soubor odstranit,
5. nový soubor přejmenovat na název původního.

Podařilo se mi vymyslet lepší řešení. `Duration`, `audiocodecid` i `videocodecid` jsou všechno čísla. `Meta` hlavička se zapisuje pomocí formátu AMF0, který číslo vždy reprezentuje 8 byty, ať má jakoukoliv hodnotu. Hlavička:

```
{
    "audiocodecid": 0,
    "videocodecid": 0,
    "duration": 0
}
```

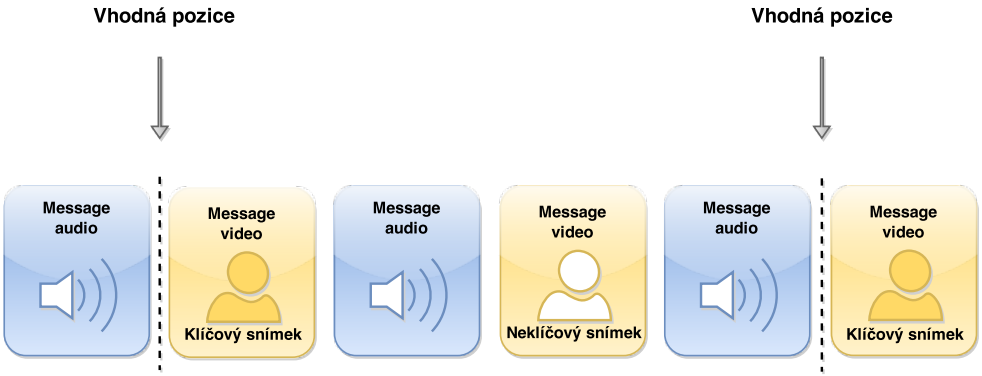
bude dlouhá úplně stejně jako např.:

```
{
    "audiocodecid": 2048,
    "videocodecid": 4,
    "duration": 62.5
}
```

Díky tomu si můžeme dovolit malý trik: do souboru zapíšeme fiktivní hlavičku s defaultními hodnotami a jakmile uživatel přestane nahrávat, známe už skutečnou délku, hlavičku přepíšeme jediným voláním funkce operačního systému. Žádné `tagy` se kopírovat nemusí.

2.6 Návrh .meta souborů

Již jsem se zmínil o problému seekování, tedy změně pozice v přehrávaném videu. Node.js dokáže při čtení přesunout ukazatel v souboru na libovolnou pozici, nicméně správnou pozici není snadné zjistit. Když například uživatel chce skočit do poloviny videa, server nemůže číst od poloviny souboru. Soubor se skládá z `tagů` různé délky. Téměř jistě by skočil dovnitř některého z `tagů` a musel by uživatele informovat o neúspěchu.



Obrázek 2.5: Vhodné pozice pro seekování.

Další problém souvisí se způsobem, jakým je komprimován obraz. Aby se zmenšila velikost dat, snímky se dělí do dvou typů: klíčové a interpolované [23]. Klíčové snímky jsou takové, které obsahují všechny potřebné informace o obraze. Z klíčového snímku dokáže dekodér získat obraz. Interpolované snímky jsou mnohem menší, neobsahují informace o celém obraze, ale pouze změny oproti předchozímu snímku. Aby mohl dekodér zrekonstruovat obraz interpolovaného snímku, potřebuje úplně všechny interpolované snímky před ním a poslední klíčový snímek. Z tohoto důvodu nemůže RTMP server začít streamovat obraz z jakékoliv pozice, ale musí začít klíčovým snímkem. Pro přehlednost zobrazuji vhodné pozice pro začátek streamování na obrázku 2.5.

Pozice klíčových snímků v souboru se dají zjistit jen sekvenčním procházením, to může být pro velké soubory velmi pomalé. Naštěstí zjistit pozice nemusí server pokaždé, když uživatel chce seekovat. Postačí, když to udělá pouze jednou a uloží si je do souboru. Přečíst pozice ze strukturovaného souboru je mnohem rychlejší.

Navrhl jsem pro server k tomuto účelu soubory s příponou `.meta`. Například videu `movie.flv` odpovídá soubor s názvem `movie.flv.meta`. Jakmile uživatel začne přehrávat video, server soubor `.meta` načte do paměti. Jestliže soubor neexistuje, musí ho nejprve vygenerovat. Mezitím, co server streamuje video uživateli, na pozadí jej celé sekvenčně projde a zapamatuje si pozice klíčových snímků. Pokud uživatel požádá o změnu pozice ve videu, server v paměti najde vhodnou pozici a přesune ukazatel v souboru.

Soubory `.meta` neobsahují jen klíčové snímky, všechny jeho vlastnosti jsou:

keyframes – Klíčové snímky videa. Ukládají se v podobě asociativního pole – klíče jsou časy video `tagů`, hodnoty jejich pozice v souboru.

meta – Metadata video souboru.

modified – Datum poslední modifikace. Pokud se od té doby video změnilo, server musí `.meta` soubor smazat a vygenerovat znovu.

Soubory se ukládají ve formátu JSON, se kterým se velmi pohodlně pracuje. Samotný Node.js poskytuje objekt JSON a jeho metody `stringify` a `parse` pro serializování a deserializování [24]. Ukázka `.meta` souboru s bohatě vyplněnými metadaty:

```
{
   "modified": 1410604030995,
   "meta": {
      "duration": 65.437,
      "width": 848,
      "height": 360,
      "videodatarate": 781.25,
      "framerate": 23.976023976023978,
      "videocodecid": 4,
      "audiodatarate": 93.7421875,
      "audiosamplerate": 44100,
      "audiosamplesize": 16,
      "stereo": true,
      "audiocodecid": 2,
      "filesize": 3137481
   },
   "keyFrames": {
      "0": 621,
      "375": 8067,
      "918": 50673,

      ...

      "50676": 2638376,
      "51593": 2725372,
      "53554": 2804971
   }
}
```

Skutečné soubory na disku jsou ukládány bez mezer a konců řádků. Tento příklad jsem zobrazil voláním metody `stringify` s parametry pro čitelnější výstup. Komprimovaná podoba je pro lidi méně srozumitelná, je však podstatně menší.

2.7 Návrh AudioVideo modulu

Úkolem posledního modulu je práce se zvukovými a obrazovými kodeky. Dokáže dekódovat zvuk a video z binárních dat nebo naopak, převádět mezi barevnými modely, měnit rozlišení snímků a další nezbytné funkce pro práci s audiem a videem, které RTMP modul neumí. AudioVideo se hodí pro rozšíření streamovacího serveru, avšak může být použit i k jiným účelům.

Z analýzy vyplynulo, že pro práci s kodeky je nejvhodnější využití Libav knihovny. Libav je psána v jazyce C, bylo tedy nutné ji implementovat do modulu a vytvořit JavaScriptové rozhraní. Většina zdrojového kódu je psána v jazycích C a C++, proto musí být modul před spuštěním zkompilován. Naštěstí ostatní moduly pracují nezávisle na AudioVideo modulu, takže pokud jeho funkce nejsou potřeba, je možné se kompilaci vyhnout. Této problematice se věnuji v kapitole Instalační příručka.

2.8 Návrh tříd AudioVideo modulu

Libav nabízí velkou řadu funkcí. Některé z nich jsou nezbytné pro tuto práci, jiné s ní však vůbec nesouvisí. Implementoval jsem pouze nezbytné součásti a to s důrazem na jednoduchost, na rozdíl od samotné knihovny, která je na použití poměrně komplikovaná.

První zjednodušení jde ruku v ruce s objektovým přístupem. Zdrojový kód v jazyce C se skládá ze struktur a samostatných funkcí. Jaké funkce struktura podporuje (přesněji řečeno jaké funkce se strukturou pracují) je nutné vždy hledat v dokumentaci, kdežto JavaScriptové objekty jasně definují jakými metodami disponují.

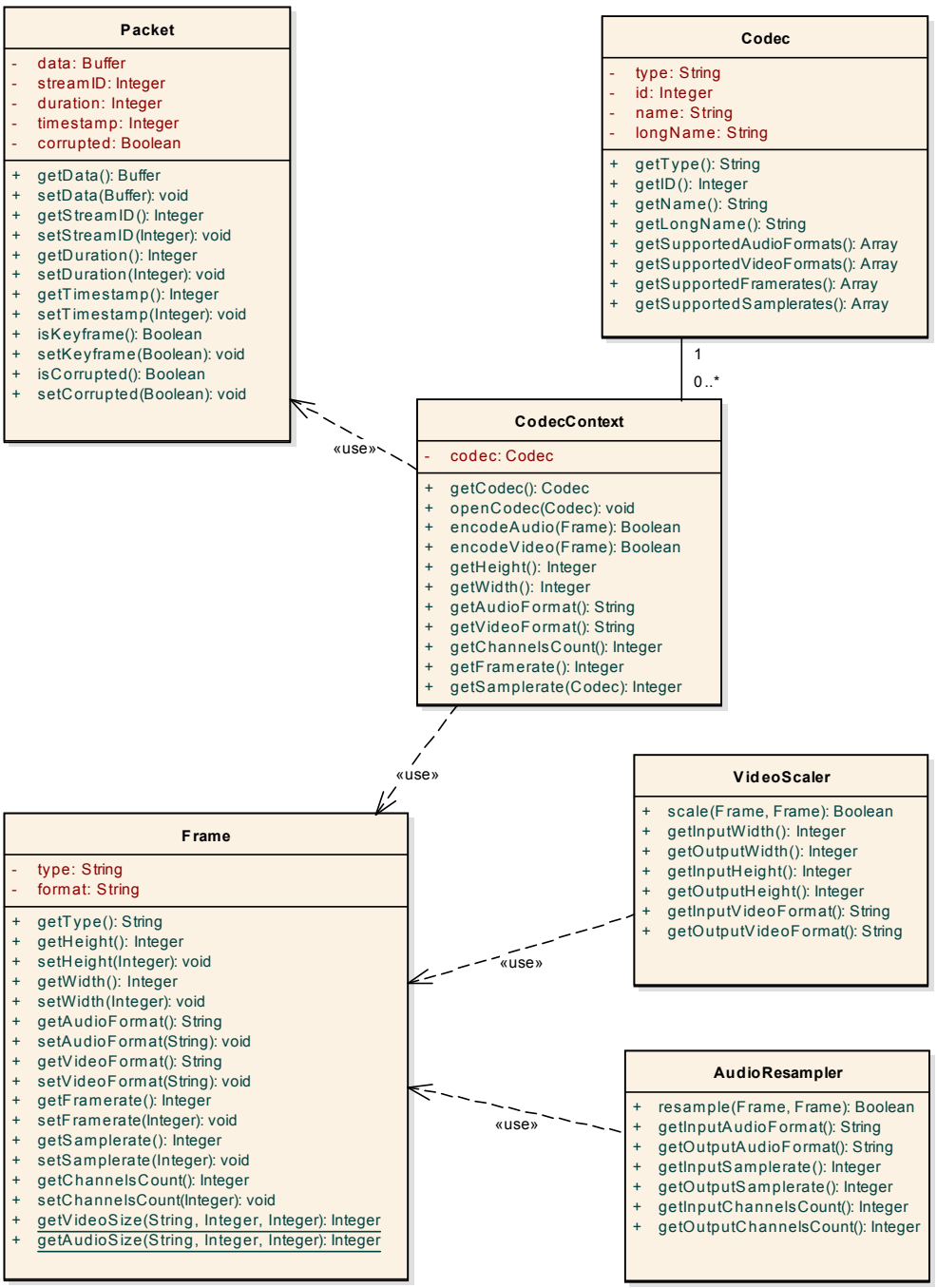
Druhé zjednodušení spočívá ve struktuře, kterou jsem navrhl. Třídy jsem se snažil vytvořit tak, aby práce s nimi nebyla složitá, ale současně knihovna neztrácela na výkonu ani univerzálnosti. Určitě se mi to z části podařilo: zápis některých úkolů v JavaScriptu je mnohem kratší než přímo volání funkcí Libav. Současně jsem nic neubral na obecnosti, kupříkladu v ukázkové aplikaci modul používám k zápisu PNG obrázků, přestože k tomuto účelu nebyl vytvořen.

Navrhl jsem několik tříd, které obalují knihovnu Libav. Volat její funkce přímo není možné, protože jsou psány v jazyce C. Pro přehlednost jsem zachoval podobnost názvů tříd:

Packet – Třída pro reprezentaci binárních dat; obaluje Libav strukturu `AVPacket`. Tato třída by mohla být nahrazena standardním JavaScriptovým objektem `Buffer`, avšak zvýšila by se výpočetní režie. Binární data by bylo nutné v téměř každé metodě kopírovat mezi `Bufferem` a `AVPacketem`.

Frame – Reprezentace obrazového snímku nebo zvukového záznamu; obaluje Libav strukturu `AVFrame`. Není přirozené, aby tentýž objekt jeden čas reprezentoval obrázek, posléze zvuk, nicméně rozdělení na audio `Frame` a video `Frame` není kvůli některým funkcím Libav možné.

Třída nabízí metody pro manipulaci s daty. Metodou `getData` je možné získat pixely obrázku, resp. zvuková hodnoty, upravit je a následně zase uložit metodou `setData`.



Obrázek 2.6: Diagram tříd AudioVideo modulu. Pro srozumitelnost je zachycena pouze základní struktura.

VideoScaler – Transformuje obrazový snímek z jednoho barevného modelu do jiného modelu (např. `YUV` na `RGB`) nebo mění rozlišení snímku (např. z 800x600px na 640x480px). Vstupem i výstupem operace je obrazový `Frame`.

AudioResampler – Transformuje zvuková data mezi různými formáty (např. float na integer 32 bitů se znaménkem) nebo mění počet kanálů (např. ze stereo na mono). Vstupem i výstupem operace je zvukový `Frame`.

Codec – Reprezentuje kodek; obaluje `AVCodec`. Kodek může být buďto `decoder` nebo `encoder` podle toho, jestli bude sloužit ke kódování nebo dekódování dat. Tento objekt slouží pouze ke čtení, nemá žádné modifikační metody. V aplikaci stačí mít pouze jednu instanci od každého kodéru nebo dekodéru. Vhodné je použít návrhový vzor *Singleton*.

CodecContext – Kodér nebo dekóder zvoleného kodeku; obaluje `AVCodecContext`. Obsahuje pomocné struktury nezbytné pro kódování a dekódování dat. Příznačnější název třídy by byl například Coder, avšak chtěl jsem zachovat jmennou konvenci použitou v Libav.

Libav podporuje mnoho formátů pro reprezentaci obrazových a zvukových dat. Obrazové formáty se liší v barevném modelu, podpoře alfa kanálu, pořadí jednotlivých složek a počtu bitů přiřazených každé složce. Zvuková data jsou přenášena jako jednorozměrné pole hodnot. Formát zvuku definuje, jak bude každá hodnota zapsána. V tabulce 2.2 předkládám nejběžnější obrazové formáty. Základní zvukové formáty obsahuje tabulka 2.3. Kompletní seznam obsahuje dokumentace `AVPixelFormat` resp. `AVSampleFormat`. V JavaScriptovém kódu oba formáty reprezentuji řetězcem.

2.9 Návrh filtrů AudioVideo modulu

AudioVideo modul je možné použít k libovolnému úkolu, avšak ve většině případů bude doplňovat modul RTMP o práci s kodeky. Abych tuto činnost zjednodušil, vytvořil jsem několik filtrů pracujících se strukturami `Message`, které generuje FLVMessageDecoder (nejvyšší filtr RTMP modulu). Filtry pracují pouze s typy `Audio` a `Video`. Podtyp `Audio` obsahuje:

```
{
    "codecID": Integer
    "stereo": Boolean
    "samplerate": Integer
    "body": Buffer
}
```

Nejzajímavější je vlastnost `body` ve které jsou kódovaná zvuková data. Ostatní vlastnosti slouží k jejich správnému dekódování. `Stereo` a `samplerate` některé

Libav konstanta	JS název	Model	Poznámka
`AV_PIX_FMT_RGBA`	`rgba`	`RGB`	1 byte na každou složku v pořadí: červená, zelená, modrá. Poslední byte reprezentuje alfa kanál.
`AV_PIX_FMT_RGB24`	`rgb24`	`RGB`	1 byte na každou složku v pořadí: červená, zelená, modrá.
`AV_PIX_FMT_RGB8`	`rgb8`	`RGB`	Červená složka: 3 bity. Zelená složka: 3 bity. Modrá složka: 2 bity.
`AV_PIX_FMT_YUYV422`	`yuyv422`	`YUV`	`Y` složka: 4 bity. `U` složka: 2 bity. `V` složka: 2 bity.

Tabulka 2.2: Formáty obrazových dat.

Tabulka 2.3: Formáty zvukových dat.

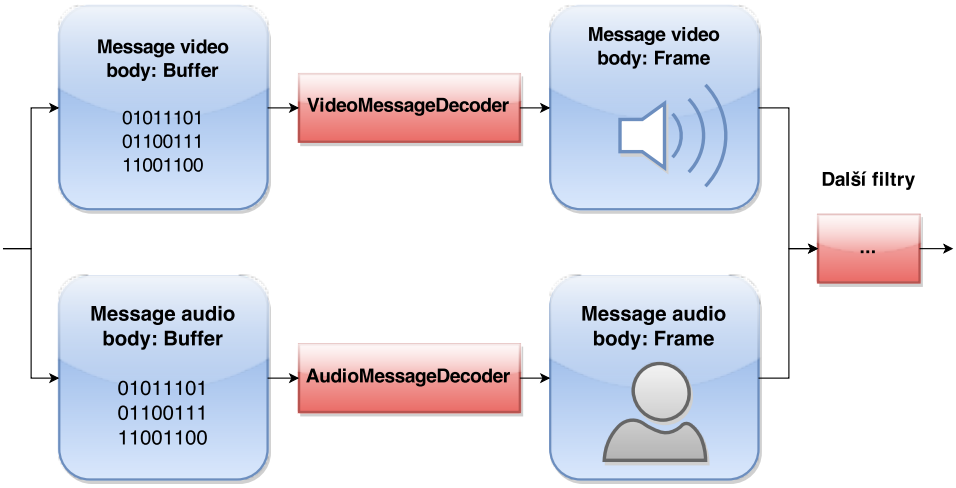
Libav konstanta	JS název	Poznámka
`AV_SAMPLE_FMT_U8`	`u8`	int 8 bitů bez znaménka
`AV_SAMPLE_FMT_S16`	`s16`	int 16 bitů se znaménkem
`AV_SAMPLE_FMT_S32`	`s32`	int 32 bitů se znaménkem
`AV_SAMPLE_FMT_FLT`	`flt`	float
`AV_SAMPLE_FMT_DBL`	`dbl`	double

kodeky nepotřebují, zatímco `codecID` je nutný vždy. Je společný také pro podtyp `Video` `Message`:

```
{
    "codecID": Integer
    "keyFrame": Boolean
    "body": Buffer
}
```

Povšimněte si, že jak `Audio`, tak `Video` uchovávají `body` jako `Buffer`, tzn. sekvenci bytů. Jejich význam má pramálo společného se zvukovými a obrazovými daty vhodnými k manipulaci. Data se musí nejprve dekódovat; k tomu jsem navrhl filtry:

VideoMessageDecoder – Dekóduje binární data (`Buffer`) na obraz (`Frame`) podle použitého kodeku. Používá se pro `Message` typu `Video`.



Obrázek 2.7: Schéma dekódovacích filtrů AudioVideo modulu.

AudioMessageDecoder – Dekóduje binární data (`Buffer`) na zvuk (`Frame`) podle použitého kodeku. Používá se pro `Message` typu `Audio`.

VideoMessageEncoder – Kóduje obraz (`Frame`) na binární data (`Buffer`) podle použitého kodeku. Používá se pro `Message` typu `Video`.

AudioMessageEncoder – Kóduje zvuk (`Frame`) na binární data (`Buffer`) podle použitého kodeku. Používá se pro `Message` typu `Audio`.

Tyto filtry aplikujeme na stream pouze jednou. Také je důležité dodržet jejich správné pořadí zachycené na diagramu 2.7. Zobrazuji v něm pouze proces dekódování dat, opačný postup je zřejmý – filtry *MessageDecoder jsou nahrazeny za odpovídají *MessageEncoder a obráceno pořadí. Posledním typem jsou filtry pro transformaci dat. Jejich vstupem i výstupem je objekt `Frame`. Mohou se vkládat v libovolném pořadí, dokonce je často potřeba tentýž typ filtru aplikovat několikrát. Stačí dodržet podmínku, aby byly umístěny mezi VideoMessageDecoder a AudioMessageEncoder, resp. AudioMessageDecoder a AudioMessageEncoder. Transformaci provádí 2 filtry:

VideoMessageScaler – Transformuje obrazová data do požadovaného obrazového formátu a rozlišení.

AudioMessageResampler – Transformuje zvuková data do požadovaného zvukového formátu a počtu kanálů.

Filtry jsou vhodným pomocníkem pro změnu rozlišení videa nebo převedení zvuku ze sterea na mono. Dále se uplatní pokud chceme upravovat data ve formátu, kterému nerozumíme nebo je to příliš komplikované. Např. video formát `rgb8` ukládá celý pixel pouze v jednom bytu. Jednotlivé barevné složky

musíme při každé manipulaci oddělit a poté zase zkombinovat binárními operacemi.

Konverzi mezi formáty bychom mohli napsat přímo v JavaScriptu. Avšak daleko jednodušší je nechat knihovnu Libav aby to provedla za nás pomocí transformačních filtrů. Libav je psána v jazyce C a zkompilovaná s mnoha výkonnostními optimalizacemi, některé klíčové funkce dokonce využívají hardwarovou akceleraci. Pravděpodobně bude mnohem rychlejší než JavaScriptový kód interpretovaný v Node.js.

Kapitola 3

Realizace

V této kapitole popisuji způsob, jakým jsem postupoval při tvorbě práce. Zaměřuji se zejména na problémy a komplikace, se kterými jsem se setkal.

3.1 Realizace RTMP modulu

RTMP modul slouží k implementaci protokolu RTMP. Obsahuje třídy pro tvorbu serveru, obsluhu klientů, streamování a publikování videí. Tvorbu modulu jsem započal analýzou modulů v repozitáři Node.js, které souvisí s problematikou. Rozhodl jsem se použít jako základ práce node-rtmpapi, který slouží ke stejnému účelu, jaký jsem očekával od vlastního modulu RTMP. Protože nepodporoval všechny typy `Message`, musel jsem jej nejprve rozšířit o typy `Audio`, `Video`, `SharedObject` a `Aggregate`. Později se ukázalo, že obsahuje také velké množství chyb.

Node.js vyniká obrovským množstvím modulů ke všem možným účelům. Vytvořit nový modul je opravdu snadné, bohužel většina zveřejněných modulů nikdy není dotažena do produkční verze. Nejinak tomu je s node-rtmpapi, jehož verze je k dnešnímu dni 0.0.1. Poslední úprava byla provedena před osmi měsíci.

Nejprve jsem se pokusil modul rozšířit o chybějící funkce a opravit v něm chyby. Nicméně se ukázalo, že bude vhodnější vytvořit úplně nový modul. Node-rtmpapi jsem použil jako základ práce. Při vývoji jsem vycházel nejen z původního modulu, ale samozřejmě z oficiální specifikace protokolu. Dokument vydaný společností Adobe však obsahuje mnoho nejasností a nepřesností:

- používání výrazu „streamID“ pro dva odlišné pojmy (`chunkStreamID` a `messageStreamID`),
- neobjasnění významů některých bytů (např. při inicializaci spojení – tzv. *handshake*),
- chybějící popis sdílených objektů (`SharedObjects`),

- chybějící popis zpráv `Aggregate`,
- jen částečný popis `Audio` a `Video` zpráv,

Marně bychom v dokumentu také hledali popis AMF dat ve zprávách `Meta`, `Command` a `SharedObject`. Další nedostatky popisuje [25]. Navíc správné kódování datových struktur je jen polovina problému, druhou je pochopení, jak je správně použít. Pro úspěšnou implementaci je nutné vědět kdy zprávy posílat, jaké mají mít vlastnosti a dodržet jejich správné pořadí. Přitom specifikace popisuje jen několik komunikačních scénářů. Z těchto důvodů jsem při tvorbě modulu hojně čerpal z ostatních Node.js modulů a také projektů:

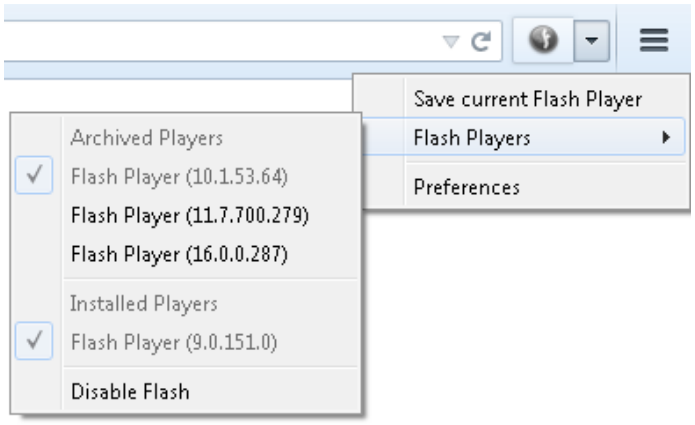
- Red5 (Java),
- Wowza (Java),
- C++ RTMP Server (C++),
- Php-rtmp-client (PHP).

Výhodou těchto projektů je otevřený zdrojový kód (v případě Wowzy jen částečně), ze kterého je patrná práce serveru. Naneštěstí žádný z nich nezaručuje správnou implementaci RTMP. Bohužel se stává, že oficiální komponenty Flash Playeru se chovají nestandardně při komunikaci s těmito servery. Jedinou spolehlivou implementaci zaručuje Adobe Media Server, který je vyvíjen přímo společností Adobe. Protože nemá zveřejněny zdrojové kódy, musel jsem jeho funkci analyzovat reverzním inženýrstvím:

1. Nainstaloval jsem Adobe Media Server.
2. Vytvořil jsem RTMP proxy server, který jsem umístil mezi AMS a Flash Player.
3. Analyzoval jsem zachycenou síťovou komunikaci.

Abych získal všechny relevantní scénáře komunikace, musel jsem nasimulovat přehrávání, publikování, práci se sdílenými objekty a vzdálené volání procedur. Chování Flash Playeru se v některých verzích změnilo. Naštěstí dokumentace ActionScriptu zobrazuje pro každou třídu, vlastnost i metodu verzi od které je dostupná a případně i verze, ve kterých se jejich chování změnilo. Abych zjistil správné komunikační scénáře, musel jsem server testovat v různých verzích Flash Playeru.

Vývojová prostředí Flashe (např. Adobe Flash Profesional CS6, který jsem používal já) dokáží nastavit verzi publikování. Slouží k tomu dialogové okno `Nastavit publikování`. V dialogu jsem upravil výstupní verzi a sledoval změny v protokolu. Bohužel nastavení má vliv pouze na podporované třídy a jejich metody ve zdrojovém kódu. Na podporované technologie (a tedy i síťovou komunikaci) vliv nemá.



Obrázek 3.1: Volba verze Flash Playeru ve Firefox pluginu Flashlight.

Abych zjistil skutečné chování jednotlivých verzí Flash Playeru, musel jsem jednu po druhé nainstalovat. Starší verze jsou dostupné na oficiálních stránkách `https://helpx.adobe.com/flash-player/kb/archived-flash-player-versions.html`. Adobe nabízí všechny Flash Playery od verze 2 až po 16, která je v tuto chvíli nejaktuálnější. Otestování jedné verze znamená nejprve odinstalování předchozí, instalace a následně restart prohlížeče. Tento postup je zdlouhavý, proto jsem hledal způsob, jak provozovat několik verzí současně.

Vyzkoušel jsem Operu, Safari, Mozillu Firefox, Internet Explorer i Google Chrome – všechny do jednoho při spuštění prohledají v počítači nainstalované verze Flash Playeru. Do svého jádra načtou a používají vždy pouze tu nejnovější. Nakonec se mi podařilo najít plugin Flashlight pro Firefox, který dokáže uložit aktuálně nainstalovanou verzi a kdykoliv ji obnovit. Umožňuje mít současně nainstalovaný libovolný počet verzí. Je sice nutné každou nainstalovat zvlášť, restartovat Firefox, uložit verzi do Flashlightu a zase ji odinstalovat, avšak pouze jednou, přepínání mezi verzemi je rychlé.

Pomocí tohoto postupu jsem mohl získat správnou síťovou komunikaci napříč různými verzemi Flash Playeru. Díky detailní analýze se mi podařilo zmapovat RTMP ve všech pluginech od těch nejstarších až po nejnovější. Vytvořil jsem server, který implementuje protokol správně. Komunikuje přesně jako Adobe Media Server, takže funguje bez problémů se všemi Flash komponentami.

3.2 Realizace AMF modulu

Protokol RTMP používá pro serializaci dat formát AMF. Mohl bych RTMP i AMF implementovat v jednom modulu, nicméně jsem shledal vhodnější je rozdělit do dvou. Protože je modul AMF zcela nezávislý, může být použit i v jiných projektech. Při tvorbě vlastního modulu jsem nejprve analyzoval

všechny moduly v repozitáři, které formát podporují. Výsledkem analýzy bylo, že nejvíce se blíží požadavkům práce node-amfutils, který jsem použil jako základ práce. Další inspiraci jsem čerpal z ostatních Node.js modulů a také z projektů:

- Red5 (Java),
- AmfPHP (PHP),
- SabreAMF (PHP),
- Php-amf3 (PHP),
- Zend_Amf (PHP).

Jejich zdrojové kódy se v serializaci některých datových typů lišily. Abych si byl jistý, že bude má implementace správná, obrátil jsem se na oficiální nástroj. Skutečně důvěryhodný zdroj je objekt ActionScriptu ByteArray. Jeho metody `read*` a `write*` slouží k serializaci a deserializaci dat. Vlastnost `objectEncoding` určuje výstupní formát – buď AMF0 nebo AMF3. Výsledné byty jsou přístupné operátory indexace `[]`, stejně jako prvky pole. Zjednodušený příklad jak je možné získat např. serializovanou podobu čísla `123` ve formátu AMF0:

```
var buffer:ByteArray=new ByteArray();

// 0 pro AMF0, 3 pro AMF3
buffer.objectEncoding=0;

buffer.writeObject(123);

for (var i:int=0; i<buffer.length; i++) {
   trace(buffer[i]);
}
```

Výstup skriptu (na jednom řádku):

```
0 64 94 192 0 0 0 0 0
```

Tzn. číslo 123 se kóduje ve formátu AMF0 byty:

```
0x00 0x40 0x5E 0xC0 0x00 0x00 0x00 0x00 0x00
```

Opakováním tohoto postupu s jinými hodnotami jsem získal zaručeně správné kódy pro všechny podporované datové typy. Některé z kódů jsem použil pro tvorbu jednotkových testů. Jsou to cenná data pro testování, zda modul funguje správně. Více o problematice popisuji v kapitole Testování.

3.3 Realizace AudioVideo modulu

AudioVideo modul obsahuje knihovnu Libav a proto musí být jako jediný psán v JavaScriptu a také C++, narozdíl od ostatních, které byly psány v čistém JavaScriptu. Zdrojové kódy v C++ je nutné nejprve zkompilovat. Úspěšné nastavení prostředí může chvíli trvat. Na internetu jsou dobré návody, ale bohužel taky spousta špatných, kvůli kterým jsem se několik dní zdržel. Podařilo se mi najít fungující návod, který popisuje správný postup krok za krokem [26].

Modul jsem vyvíjel v nástroji Microsoft Visual C++, který dokáže kompilovat moduly do požadovaného formátu interpretem Node.js. Je nutné nastavit výstupní formát jako dynamickou knihovnu s příponou `.node`. Abychom mohli volat funkce interpretu, musíme nalinkovat hlavičkové soubory Node.js a V8.

Modul obaluje funkce knihovny Libav. Jsou dva způsoby, jak knihovnu propojit s modulem:

1. Stáhnout zdrojové kódy Libav a také všech projektů, které knihovna využívá. Zdrojové kódy vložit do modulu a následně zkompilovat.

2. Využít již zkompilované zdrojové kódy v podobě dynamických knihoven. Do projektu se pouze nalinkují hlavičkové soubory a modul se zkompiluje zvlášť. Soubory knihoven musí být distribuovány současně s modulem.

Zvolil jsem druhou možnost, díky tomu jsem nemusel řešit případné problémy při kompilaci. Vyvíjel jsem v nástroji Microsoft Visual C++ v operačním systému Microsoft Windows 7 a proto jsem použil knihovny dostupné z `http://builds.libav.org/windows/`. Knihovny jsou také ke stažení pro jiné operační systémy.

Vytvořený modul dokáže pracovat pouze s audio a video kodeky. Multimediální kontejnery nepodporuje. Přestože knihovna Libav nabízí mnohem více možností, musel bych navrhnout a implementovat výrazně složitější strukturu modulu. Přitom pro účely této práce by jediným přínosem byla podpora jiných multimediálních kontejnerů než FLV. Současně rozhraní pro práci s kodeky, ke kterému je modul primárně určen, by se výrazně zkomplikovalo.

Kapitola 4

Použití serveru

V této kapitole popisuji způsob, jak se streamovacím serverem pracovat. Kvalitu videa a zvuku ovlivňuje obrovským způsobem vhodné nastavení, proto věnuji pozornost všem nastavitelným parametrům a jejich vlivu na velikost dat. Zpravidla platí, že čím kvalitnější přenášíme obraz a zvuk, tím větší datový tok vyžaduje. Množství přenášených dat neovlivňuje pouze kvalitu pozorovanou uživateli, ale také zatížení streamovacího serveru.

4.1 Vytvoření serveru

Na straně serveru musíme nejprve vytvořit instanci třídy NetServer. Objekt generuje několik událostí, které je vhodné obsloužit. JavaScript pracuje asynchronně, a proto je celý kód protkaný událostmi a callbacky. Základní struktura kódu, která úspěšně přijme spojení klienta:

```
/* SERVER (Node.js) */

// importování RTMP modulu
var RTMP=require("./RTMP/");

// vytvoření instance NetServer
var server=new RTMP.NetServer({
    port: 1935,
    host: "localhost"
});

// obsloužení klientského spojení
server.on("connection", function(connection) {

    connection.on("connect", function(callback, data) {
        // povolit navázání spojení
        callback(true);
    });
```

```
    connection.on("stream", function(stream) {
        // obsloužení událostí: (on)play, (on)publish, (on)close....
    });

    // obsloužení dalších událostí connectionu
    // [...]
});

// obsloužení dalších událostí serveru
// [...]

// spuštění serveru
server.open();
```

Server naslouchá na standardním portu 1935 na příchozí RTMP spojení. Vytvoření serveru bude úspěšné pokud je síťový port volný a Node.js má oprávnění na něm vytvořit serverový socket. V opačném případě server vygeneruje událost `(on)error`. Ve skutečné aplikaci je vhodné obsloužit všechny události serveru.

4.2 Připojení k serveru

K serveru se mohou připojovat libovolní RTMP klienti, nejčastěji však Flash Player umístěný uvnitř internetové stránky. Flash nabízí rozhraní pro snadnou komunikaci se serverem a práci s multimédii. K tomu jsou určeny třídy ActionScriptu:

flash.net.NetConnection – Obousměrné spojení mezi serverem a klientem prostřednictvím protokolu RTMP. Nabízí rozhraní pro přenos multimediálních dat prostřednictvím třídy NetStream, vzdálené volání procedur (RMI) a správu sdílených objektů.

flash.net.NetStream – Stream pro přenos audio, video a strukturovaných dat mezi klientem a serverem. Reprezentuje multimediální proud dat, např. video nebo živé vysílání. Slouží ke streamování a publikování dat. V jednu chvíli však dokáže buď streamovat nebo publikovat, nikdy současně obojí.

NetStream se připojuje k NetConnection. K jednomu spojení může být připojený libovolný počet streamů.

flash.media.Camera – Rozhraní pro zachycení videa z webkamery uživatele. Připojením webkamery ke streamu publikujeme video data na server. Její nastavení ovlivní kvalitu a velikost přenášeného videa.

flash.media.Microphone – Rozhraní pro zachycení zvuku z mikrofonu uživatele. Připojením mikrofonu ke streamu publikujeme audio data na server. Jeho nastavení ovlivní kvalitu a velikost přenášeného zvuku.

flash.media.Video – komponenta pro zobrazení přehrávaného videa. Jeho parametry jsou šířka a výška, nicméně toto nastavení nemá vliv na rozměry přenášeného videa, nýbrž pouze na způsob, jak se video zobrazí uživateli.

V následujících sekcích popíši práci s těmito třídami a také jejich vhodné nastavení. U každé ukázky zdrojového kódu pro přehlednost uvádím, jestli je psán ve Flashi na klientské straně nebo v JavaScriptu na serverové straně. Nejprve je nutné vytvořit spojení s RTMP serverem, které zobrazuje následující kód:

```
/* KLIENT (Flash Player) */

var connection:NetConnection=new NetConnection();

// vlastnosti spojení
connection.objectEncoding=ObjectEncoding.AMF0;
// [...]

// navázání spojení
connection.connect("rtmp://localhost/appName");
```

Jedná se o funkční, nicméně pouze základní kód. Je vhodné alespoň obsloužit události třídy NetConnection, které informují o úspěchu nebo neúspěchu navázání spojení. Dále popisuji, jak spojení využít k přenášení multimediálních dat mezi Flashem a serverem. Neopomíjím popis faktorů, které mají vliv na zatížení serveru a kvalitu dat.

4.3 Publikování videa a zvuku

Zvuk a video je publikováno prostřednictvím NetStreamu. K objektu je nejprve nutné připojit webkameru nebo mikrofon, ze kterých chceme data zachycovat. Metoda `publish` spustí publikování. Předpokládejme již navázané spojení, potom je základní kostra pro publikování dat na server:

```
/* KLIENT (Flash Player) */

// vlastnosti videa
var camera:Camera=Camera.getCamera();
camera.setMode(320, 240, 25);
camera.setKeyFrameInterval(25);
// [...]
```

```
// vlastnosti zvuku
var microphone:Microphone=Microphone.getMicrophone();
microphone.codec=SoundCodec.NELLYMOSER;
// [...]

// vytvoření multimediálního streamu
var stream:NetStream=new NetStream(connection);
// připojení videa ke streamu
stream.attachCamera(camera);
// připojení audia ke streamu
stream.attachAudio(microphone);

// publikování streamu s názvem StreamName živě na server
stream.publish("StreamName", "live");
```

Voláním metody `publish` vyžádáme po Flash Playeru spuštění přenosu. Flash pošle příkaz pro žádost o publikování a aniž by čekal na odpověď, okamžitě začíná zapisovat data do protokolu. RTMP modul na straně serveru zpracuje žádost a vygeneruje událost `(on)publish`. Nejjednodušší způsob zpracování události je uložení publikovaných dat do souboru:

```
/* SERVER (Node.js) */

stream.on("publish", function(callback, name, type) {

    // vytvoří FileWriter pro čtení FLV dat ze souboru
    var writer=new RTMP.FileWriter(filename);

    // událost generovaná pokud se soubor podaří otevřít
    writer.on("open", function() {

        // povolí publikování dat
        // současně vytvoří Reader pro čtení
        var reader=callback(true);

        // zde se mohou na data aplikovat různé filtry
        // [...]

        // ukládání dat do souboru
        reader.pipe(writer);

        // spustí čtení publikovaných dat
        reader.start();
    });

});
```

Publikovaná data mohou být ukládána současně do několika souborů a ještě streamována jiným uživatelům. Pokud bychom chtěli data nejprve upravit, označil jsem místo, kde je vhodné vložit filtry. Data můžeme upravovat vlastními filtry nebo využít těch z modulu AudioVideo.

4.4 Vlastnosti publikovaného videa

Zdrojem publikovaných obrazových dat je webkamera, proto jejich kvalitu a velikost může ovlivnit pouze Flash na straně klienta [27]. V této sekci se věnuji vhodnému nastavení parametrů. Jedním z největších faktorů je volba kodeku, který určuje vlastnost NetStreamu `videoStreamSettings`. Přestože Flash dokáže přehrávat v mnoha video kodecích, při výběru kodeku pro publikování máme pouze dvě možnosti – viz. tabulka 4.1.

Další vlastnosti se týkají přímo publikovaného obrazu. Nastavujeme je prostřednictvím metod objektu Camera:

setMode(width:int, height:int, fps:Number):void – Základní nastavení obrazu nastavuje metoda `setMode`. První dva parametry definují rozlišení snímků, třetí parametr určuje frekvenci snímků:

width – šířka obrazu,

height – výška obrazu,

fps – počet snímků za sekundu.

Intuitivně platí, že čím je rozlišení snímku a počet snímků za sekundu větší, tím větší jsou také kódovaná data. Pokud chceme dosáhnout plynulého obrazu, neměl by být počet snímků menší než 25. Nicméně jestliže Flash Player usoudí, že je příliš vytížen procesor nebo je zahlceno síťové spojení, může některé snímky zahazovat navzdory nastavené frekvenci.

setKeyFrameInterval(keyFrameInterval:int):void – Nastavení četnosti klíčových snímků. Hodnota udává, které video snímky se přenesou jako klíčové a které jako interpolované. Interpolované snímky jsou zpravidla mnohem menší, neobsahují úplnou informaci o obraze, ale vypočítají se na základě předchozích snímků. Přináší tudíž vyšší režii při manipulaci s videem, této problematice se věnuji v kapitole Návrh.

Přípustné hodnoty jsou 1 až 48. Hodnota 1 znamená, že každý snímek je klíčový. Hodnota 48 znamená, že každý 48. snímek je klíčový, atd. Vyšší

Název	Hodnota	Konstanta	Poznámka
H.264	`H264Avc`	`VideoCodec.H264AVC`	
Sorenson	`Sorenson`	`VideoCodec.SORENSON`	výchozí

Tabulka 4.1: Volba video kodeku.

hodnota odpovídá menšímu procentu klíčových snímků, tedy menšímu datovému toku. Naopak menší hodnota znamená četnější klíčové snímky, tedy větší datový tok.

Pokud je klient připojen s opravdu špatným síťovým připojením, mělo by být procento klíčových snímků co nejmenší a to i na úkor, že bude muset klíčové snímky do obrazu vkládat server.

setQuality(bandwidth:int, quality:int):void – Nastavení kvality obrazu. Volit můžeme podle dvou hledisek: maximálního datového toku a míry komprese. Slouží k tomu parametry `bandwidth` a `quality`:

bandwidth – Maximální přenosová šířka pásma v bytech za sekundu. Omezení platí pouze pro video data, nezapočítávají se data nutná k zákódování a přenesení obrazu protokolem. Stejně tak se nezapočítává režie TCP. Výchozí hodnota je 16384. Hodnota 0 znamená žádné omezení, šířka pásma se zcela přizpůsobí nastavené kvalitě.

quality – Kvalita videa, tedy míra komprese. Udává se jako celé čísla v rozsahu od 1 (nejvyšší kompresní stupeň, nejmenší kvalita) do 100 (žádná komprese, nejlepší kvalita). Hodnota 0 znamená přizpůsobení kompresního poměru nastavené šířce pásma.

Metoda setQuality může být volána kdykoliv v průběhu publikování dat. Vhodně napsaná Flash aplikace může redukovat množství posílaných dat, aby se nezahltilo síťové spojení. V průběhu publikování může detekovat aktuální stav linky a přizpůsobovat mu kvalitu videa.

Objekt Camera dokáže identifikovat, zda se obraz změnil. Metodou `setMotionLevel` můžeme definovat minimální změnu a čas, která je pro aplikaci zajímavá:

setMotionLevel(motionLevel:int, timeout:int = 2000):void

Přesněji řečeno metoda porovnává procentuální změnu aktuálního snímku oproti předchozímu. Bohužel, i v případě dvou naprosto identických snímků budou oba poslány serveru. Nastavení nemá vliv na posílaná video data. Metoda pouze definuje zda a za jakých okolností se generuje událost `ActivityEvent.ACTIVITY` [28].

4.5 Vlastnosti publikovaného zvuku

Zdrojem publikovaných zvukových dat je mikrofon uživatele, proto jejich kvalitu a velikost může ovlivnit pouze Flash [29]. Nastavením objektu Microphone měníme formát zachycených audio dat. Slouží k tomu vlastnosti:

codec – Volba zvukového kodeku. Na výběr máme ze čtyř hodnot, jak ukazuje tabulka 4.2. Nejsou podporovány všechny kodeky, které podporuje Flash pro přehrávání videa. Na druhé straně je to větší výběr kodeků, než podporuje pro publikování videa.

Název	Hodnota	Konstanta	Poznámka
Nellymoser	`NellyMoser`	`SoundCodec.NELLYMOSER`	výchozí
PCM A-law	`pcma`	`SoundCodec.PCMA`	
PCM μ-law	`pcmu`	`SoundCodec.PCMU`	
Speex	`Speex`	`SoundCodec.SPEEX`	

Tabulka 4.2: Volba zvukového kodeku.

encodeQuality – Maximální přenosová šířka pásma v bytech za sekundu. Vlastnost dostupná pouze pro kodek Speex. Možné hodnoty předkládá tabulka 4.3. Kvalita kódování zvuku odpovídá míře ztrátové komprese. Čím je vyšší hodnota, tím lepší bude zachována kvalita zvuku a obráceně.

Stejně jako kvalitu videa, tak i kvalitu zvuku může aplikace vhodně přizpůsobovat aktuálním podmínkám síťového spojení.

Hodnota	Datový tok	Poznámka
0	3.95 kb/s	
1	5.75 kb/s	
2	7.75 kb/s	
3	9.80 kb/s	
4	12.8 kb/s	
5	16.8 kb/s	
6	20.0 kb/s	výchozí
7	23.8 kb/s	
8	27.8 kb/s	
9	34.2 kb/s	
10	42.2 kb/s	

Tabulka 4.3: Volba kvality řeči.

Tabulka 4.4: Volba vzorkovací frekvence zvuku.

Hodnota	Frekvence	Poznámka
44	44,100 Hz	
22	22,050 Hz	
11	11,025 Hz	
8	8,000 Hz	výchozí
5	5,512 Hz	

rate – Frekvence v kHz, s níž mikrofon zachycuje zvuk. Výchozí hodnota je 8 kHz. Jestliže ji mikrofon nepodporuje, použije se první vyšší, která je podporovaná – obvykle 11 kHz. Čím je frekvence vyšší, tím více je naměřeno hodnot a tím větší jsou zvuková data.

framesPerPacket – Počet audio snímků přenesených v jednom RTMP `tagu`. Vlastnost dostupná pouze pro kodek Speex. Výchozí hodnota je 2, tedy 2 zvukové snímky v jednom `tagu`.

S rostoucím počtem snímků se částečně snižuje množství přenesených dat, na druhou stranu zvyšuje zpoždění mezi odesláním a přijetím audio dat. Pokud by byla hodnota nastavena např. na 10, znamená to, že 9 bloků bude čekat na nahrání posledního, než budou odeslány serveru. V řeči čísel to znamená, že první snímek bude čekat na odeslání 180 milisekund (9 x 20 milisekund).

Objekt Microphone dokáže rozpoznat, zda mikrofon zaznamenal nějaké zvuky [30]. Narozdíl od objektu Camera dokáže redukovat množství odesílaných dat. Jestliže není zachycen žádný zvuk, jednoduše se neodešlou žádná data. Hranice mezi zvukem a nezajímavým šumem nastavuje metoda `setSilenceLevel`:

setSilenceLevel(silenceLevel:Number, timeout:int = -1):void – zvolí minimální hladinu zvuku, která je pro aplikaci zajímavá. Vše ostatní považuje za šum a nebude nahrávat. Metoda umožňuje snížit datový tok pomocí vhodné volby parametrů:

silenceLevel – hranice hlasitosti zvuku v rozsahu od 0 do 100.

timeout – požadovaná délka šumu.

4.6 Přehrávání videa a zvuku

Již jsem zmínil, že NetStream přenáší data oběma směry. Používá se nejen k publikování, ale také přehrávání dat. Připojení provádí metoda `play`, jejímž prvním parametrem je název videa. Jestliže již bylo navázáno spojení, pak je základní kostra pro přehrání videa ze serveru:

```
/* KLIENT (Flash Player) */

// vytvoření multimediálního streamu
var stream:NetStream=new NetStream(connection);

// zobrazení video dat uživateli
var video:Video=new Video();
addChild(video);
video.attachNetStream(stream);

// požádat server o přehrání streamu s názvem StreamName
stream.play("StreamName");
```

Přehrávání zahajuje vždy klientský Flash script. Obvykle uživatel klikne na ikonu `play` a ActionScript pošle serveru přes síťový socket žádost o připojení ke streamu. Stejný postup se používá jak pro přehrání souboru, tak připojení k živému vysílání. Volba streamu je čistě v rukou serveru. Jak by mohl vypadat zdrojový kód, včetně volby mezi živým streamem a videem ze souboru:

```
/* SERVER (Node.js) */

stream.on("play", function(callback, name) {

    // pokud existuje živý stream s tímto názvem
    if (name in liveStreams) {

        // potvrdím klientovi úspěšné připojení ke streamu
        // současně vytvoří Writer pro zápis dat
        var writer=callback(true);

        // streamování dat z živého streamu
        liveStreams[name].pipe(writer);

    }
    else {

        // vytvoří FileReader pro čtení FLV dat ze souboru
        // objekt podporuje seekování
        var reader=new RTMP.SeekableFileReader(filename);
```

```
        // událost generovaná pokud se soubor podaří otevřít
        reader.on("open", function() {

                // povolí publikování dat
                // současně vytvoří Writer pro zápis dat
                var writer=callback(true);

                // streamování dat ze souboru
                reader.pipe(writer);

                // zde se mohou na data aplikovat různé filtry
                // [...]

                // zahájí čtení dat ze souboru
                reader.start();

        });

});
```

V ukázce by se měla správně ošetřit větev, ve které se soubor nepodaří otevřít. Pokud stream neexistuje nebo uživatel nemá oprávnění k jeho sledování, měl by server operaci `play` odmítnut s popisem chyby.

Jak je z ukázek patrné, vlastnosti streamovaných dat může ovlivnit pouze server. ActionScript umí nastavit rozměry videa, hlasitost zvuku, aplikovat různé filtry, avšak všechny transformace probíhají pouze na straně klienta. Neprojeví se v přenášených datech. V dalších sekcích předkládám způsoby, jakými kvalitu a velikost videa upravovat na straně serveru. Pro jejich funkci je nutné mít nainstalován také modul AudioVideo.

4.7 Vlastnosti přehrávaného videa

Zdrojem přehrávaných dat jsou obvykle video soubory, proto je jejich kvalita a velikost pouze v rukou streamovacího serveru. Jestliže potřebuje server upravit vlastnosti video dat, musí je nejprve třídou VideoMessageDecoder dekódovat. Pokud je to potřeba, data upraví a následně je třídou VideoMessageEncoder zakóduje s novými parametry. Největší vliv na kvalitu a datový tok mají parametry:

codec – Obrazový kodek. Flash dokáže video publikovat pouze v kodecích: H.264 a Sorenson. Naštěstí pro přehrávání jich podporuje mnohem více. Tabulka 4.5 obsahuje všech 9 kodeků [31][22]. Sloupec `Kodek` odpovídá názvu používaném modulem AudioVideo, `ID` je identifikátor použitý protokolem RTMP.

Tabulka 4.5: Seznam všech video kodeků.

ID	Název	Kodek	Poznámka
1	Nekomprimovaná data	`rawvideo`	Formát `rgb24`.
2	Sorenson Spark	`flv1`	
3	Screen Video	`flashsv`	
4	On2 VP6	`vp6f`	Pouze dekódování.
5	On2 VP6 Alpha	`vp6a`	Pouze dekódování.
6	ScreenVideo 2	`flashsv2`	
7	H.264	`h264`	
8	H.263	`h263`	Pouze rozlišení obrazu 176x144, 352x288, 704x576, a 1408x1152 px.
9	MPEG4	`mpeg4`	

framerate – Vlastností `framerate` určujeme počet snímků za sekundu. Hodnotu nemá smysl zvětšovat, jen bychom do videa vkládali redundantní informace, některé snímky by se opakovaly. Snižování hodnoty způsobí zahazování některých snímků a přečasování ostatních. Pokud je k tomu důvod, můžeme tento proces dělat také manuálně JavaScriptem.

keyFrameInterval – Četnost klíčových snímků. Platí pro ni totéž, jako pro `keyFrameInterval` v ActionScriptu. Vlastnost udává, které video snímky se přenesou jako klíčové a které jako interpolované. Hodnota 1 znamená, že každý snímek je klíčový. Hodnota 25 znamená, že každý 25. snímek je klíčový, atd. V JavaScriptu můžeme totéž dělat voláním metody `setKeyframe` objektu Frame.

bitrate – Průměrný počet bitů použitých pro kódování jedné sekundy videa. Platí, že čím více bitů je pro snímky použito, tím je obraz kvalitnější. S parametrem je vhodné experimentovat, dokud nebude dosaženo optimálního poměru mezi kvalitou a velikostí dat. S parametrem `bitrate` souvisí `minBitrate` (minimální `bitrate`) a `maxBitrate` (maximální `bitrate`).

Poslední vlastností ovlivňující datový tok je rozlišení videa. Velikost snímků můžeme upravit vlastním algoritmem v JavaScriptu nebo využít objekt VideoMessageScaler, který bude pravděpodobně mnohem rychlejší.

4.8 Vlastnosti přehrávaného zvuku

Aby mohl streamovací server upravovat zvuková data, musí je nejprve třídou AudioMessageEncoder dekódovat a následně třídou AudioMessageDecoder zakódovat. Parametry pro kódování jsou:

codec – Volba zvukového kodeku. Flash dokáže publikovat zvuk v kodecích: Nellymoser, PCM A-law, PCM μ-law a Speex. Pro přehrávání můžeme navíc vybírat z dalších 7 kodeků nebo data posílat nekomprimovaná v surové podobě [31][22]. Tabulka 4.6 obsahuje všechny podporované kodeky včetně identifikátoru použitého v RTMP (sloupec `ID`) a názvu v AudioVideo modulu (sloupec `AudioVideo`).

birate – Vlastnost `bitrate` ovlivňuje průměrný datový tok jak videa, tak zvuku. Hodnota se udává v počtech bitů za sekundu. Minimální, resp. maximální, hranici určuje `minBitrate`, resp. `maxBitrate`.

Velikost zvukových dat můžeme zmenšit ještě před zakódováním, když ze stereo uděláme mono. Převádět zvuk z mono na stereo nemá smysl. Změnit počet kanálů můžeme prostřednictvím objektu AudioMessageResampler. Algoritmus můžeme zapsat také v JavaScriptu, avšak AudioMessageResampler využívá hardwarovou akceleraci, takže bude pravděpodobně mnohem rychlejší.

ID	Kodek	AudioVideo
0	Nekomprimovaná data	`pcm_u8`
1	ADPCM	`adpcm_swf`
2	MP3	`mp3`
3	PCM le	`pcm_s16le`
4	Nellymoser (16kHz mono)	`nellymoser`
5	Nellymoser (8kHz mono)	`nellymoser`
6	Nellymoser	`nellymoser`
7	PCM A-law	`pcm_alaw`
8	PCM μ-law	`pcm_mulaw`
10	AAC	`aac`
11	Speex	`libspeex`

Tabulka 4.6: Seznam všech audio kodeků.

Kapitola 5

Dokumentace

Na přiloženém CD je dokumentace všech JavaScriptových souborů pomocí nástroje JSDoc, který se v Node.js běžně používá. Mezi jeho hlavní výhody patří výstup v podobě HTML stránek a standardní syntaxe, velmi podobná nástroji JavaDoc pro dokumentaci zdrojových kódu psaných v Javě. Pro dokumentaci JavaScriptu se hodí zejména anotace:

@module – Definice modulu.

@callback – Definice callback funkce.

@event – Definice události.

@emits – Generátor události.

@listens – Posluchač události.

@return – Návratová hodnota funkce. Zápis JavaScriptové funkce neobsahuje datový typ návratové hodnoty, proto je vhodné ji doplnit manuálně.

@throws – Výjimka, kterou funkce vrhá. Možno použít také alias @exception. Ze zápisu funkce není možné rozpoznat výjimky, které vrhá, proto je vhodné je taktéž doplnit ručně.

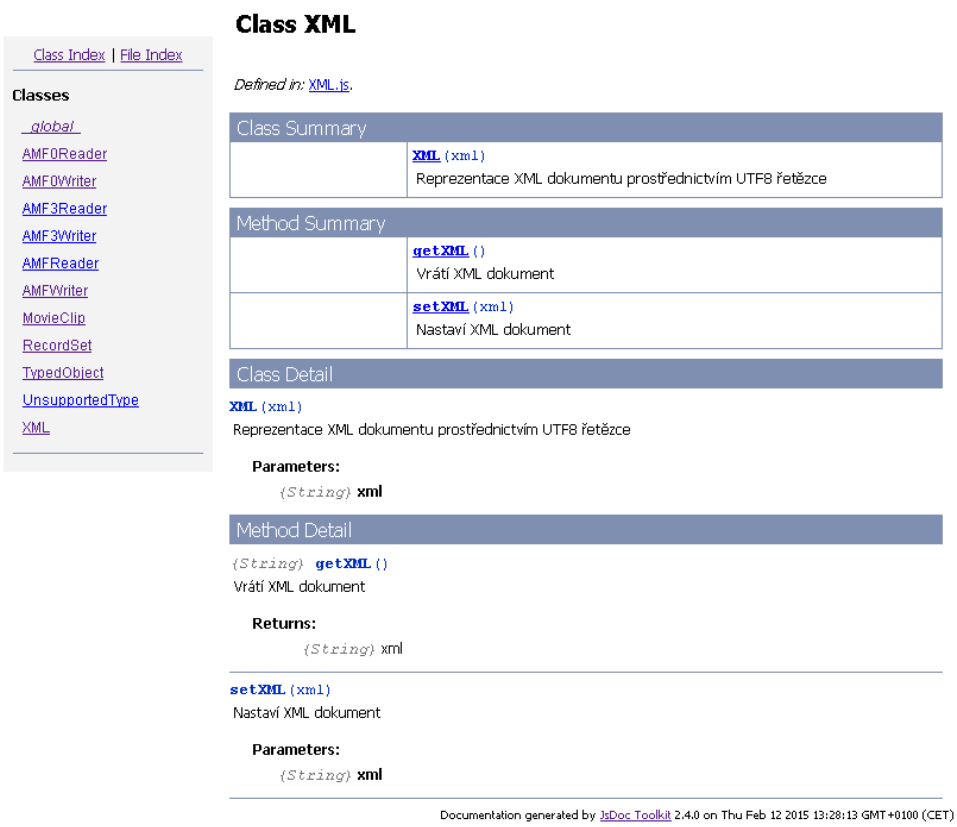
@param – Vstupní parametr funkce. Ani jejich datový typ není možné definovat v JavaScriptu. Zápis je vhodné opatřit také komentářem, k čemu parametr slouží.

@constant – Definice konstanty. JavaScript konstanty nepodporuje. Anotace umožňuje do dokumentace poznamenat, které proměnné mají být pouze pro čtení.

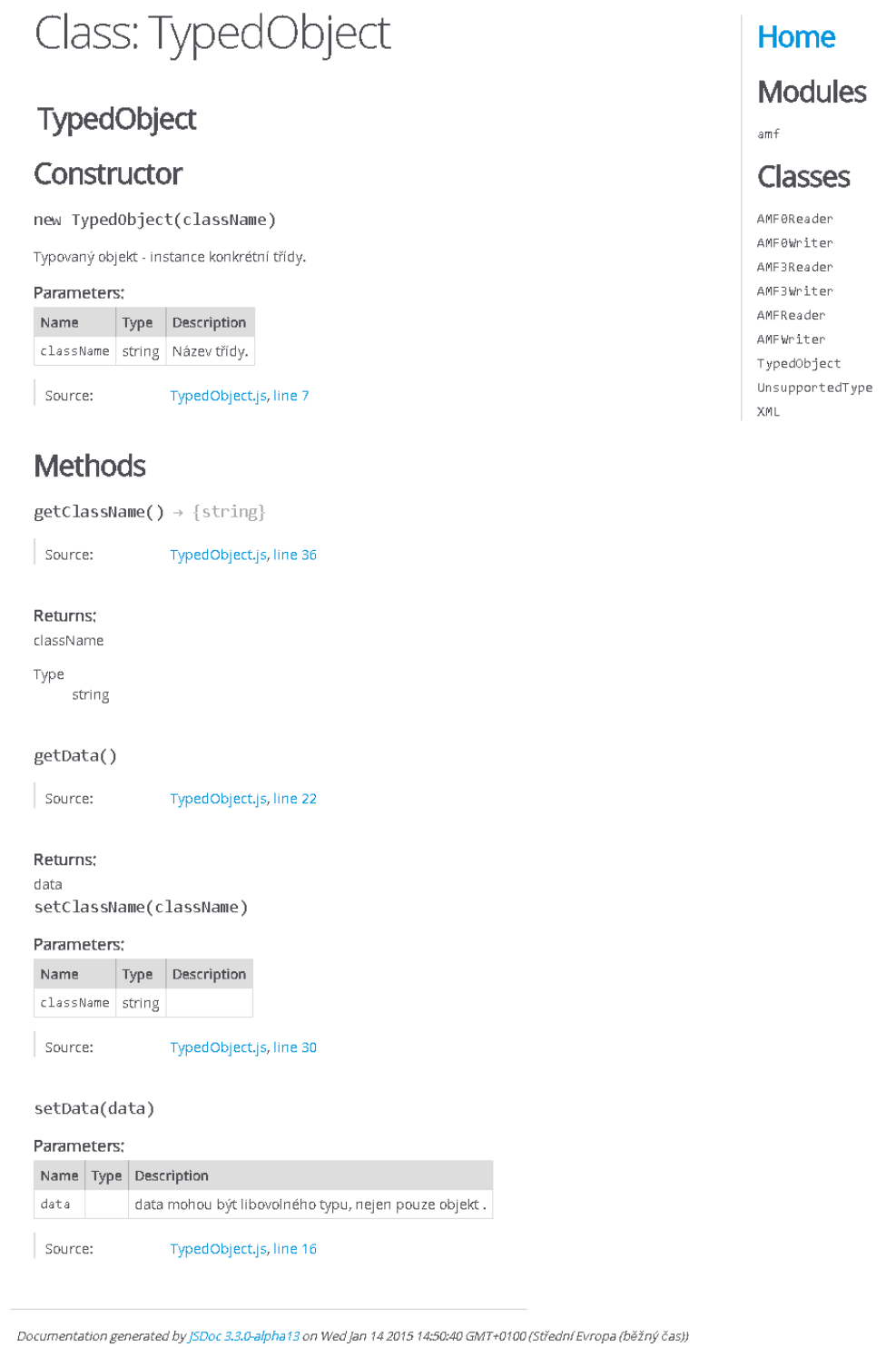
Příklad anotací pro deklaraci konstruktoru:

```
/**
* Dekodér AMF0 dat.
* @constructor
* @param {Buffer} data
* @param {int} offset - počáteční pozice
* @param {int} length - délka dat
* @emits AMFReader#close
*/
```

Nejnovější verze JSDoc 3.3.0 zavádí podporu pro `@module` a další užitečné funkce pro Node.js. Bohužel, dle mého názoru, není výstup nejnovější verze tak srozumitelný, jako verze JSDoc 2.4.0. Z tohoto důvodu jsem se rozhodl generovat dokumentaci v obou verzích. Srovnání výstupu verzí JSDoc 2.4.0 a JSDoc 3.3.0 je možné na obrázcích 5.1 a 5.2. Na CD je umístěn soubor `/doc/generate.bat`, který vygeneruje dokumentaci v obou formátech.



Obrázek 5.1: Dokumentace generovaná programem JSDoc 2.4.0.



Class: TypedObject

TypedObject

Constructor

`new TypedObject(className)`

Typovaný objekt - instance konkrétní třídy.

Parameters:

Name	Type	Description
`className`	string	Název třídy.

Source: TypedObject.js, line 7

Methods

`getClassName()` → {string}

Source: TypedObject.js, line 36

Returns:

className

Type
string

`getData()`

Source: TypedObject.js, line 22

Returns:

data

`setClassName(className)`

Parameters:

Name	Type	Description
`className`	string	

Source: TypedObject.js, line 30

`setData(data)`

Parameters:

Name	Type	Description
`data`		data mohou být libovolného typu, nejen pouze objekt .

Source: TypedObject.js, line 16

Home

Modules

amf

Classes

AMF0Reader
AMF0Writer
AMF3Reader
AMF3Writer
AMFReader
AMFWriter
TypedObject
UnsupportedType
XML

Documentation generated by JSDoc 3.3.0-alpha13 on Wed Jan 14 2015 14:50:40 GMT+0100 (Střední Evropa (běžný čas))

Obrázek 5.2: Dokumentace generovaná programem JSDoc 3.3.0.

Kapitola 6

Testování

Testování softwaru je jeden ze způsobů, jak zajistit kvalitu produktu. Cílem testování není pouze odhalit chyby. Zdrojový kód této práce jsem testoval pomocí dvou typů testů:

Jednotkové testy – Jednotkové testování, anglicky *unit testing*, ověřuje funkčnost základních jednotek. Při testování jsem za nejmenší jednotku považoval metody tříd. Test spustí metodu se vstupními parametry a kontroluje, že vrátí požadovaný výstup, příp. očekává výjimku. Jednotkové testy musí být rychlé, aby mohly být spuštěny při každé větší úpravě zdrojového kódu.

Integrační testy – Integrační testy ověřují vzájemnou spolupráci systémových částí. Kombinují třídy do větších celků a testují správný průběh procesů. Testy jsem vytvářel tak, aby prováděly běžné činnosti modulů. Každý test dokáže odpovědět na otázku, zda celek pracuje nebo nepracuje správně.

Pro tvorbu jednotkových a integračních testů existuje v Node.js spousta nástrojů. Vybral jsem framework Mocha.js a Should.js, které jsou jedny z nejpoužívanějších. Programování v JavaScriptu je jiné než v Javě, PHP nebo C++, proto ve stručnosti popíšu, jak testovací nástroje pracují. Mocha je podle oficiálních stránek testovací framework dělající testování jednoduché a zábavné. Běží v Node.js, ale může být spuštěn taky v internetovém prohlížeči. Zaměřuje se na asynchronní testování, protože většina kódu JavaScriptu je psána pomocí callbacků. Samozřejmě dokáže testovat synchronní kód, měřit dobu běhu skriptu a další užitečné funkce, které překračují působnost této práce.

Should se zaměřuje na porovnání hodnot. Dokáže samozřejmě porovnávat na základě jednoduchých operátorů, ale nejvíce programátorova času ušetří v případě složitějších testování:

- hluboké a mělké kopie objektů,
- vlastností objektů,
- přítomnosti prvků v poli,
- čísel na základě intervalů,
- řetězců regulárními výrazy,
- vrhaných výjimek.

Kombinace Mocha.js a Should.js tvoří úžasný nástroj. Při správném použití testy dokáží pomáhat s dokumentací zdrojového kódu. Ilustruje to velmi jednoduchá ukázka obsahujícího chybu:

```
describe("kalkulačka", function(){
    it("sečte čísla", function(){
        (5+4).should.eql(10);
    })
});
```

Testy se vkládají do adresáře test a spouští z příkazové řádky příkazem `mocha`. Výsledek testu zachycuje obrázek 6.1. Hlášení je velice podrobné, nabízí všechny důležité informace. Chce to trochu cviku, ale po chvíli je hledání zdroje chyby obvykle otázkou okamžiku.

Testoval jsem nejprve každý modul samostatně pomocí jednotkových a integračních testů a posléze jako celek. Níže popisuji detaily ke každému modulu:

RTMP modul – Testovací případy vychází z definice RTMP protokolu. Ověřují správné kódování a dekódování všech struktur a také, že se si server dokáže poradit s nestandardní sekvencí zpráv.

AMF modul – Testy ověřují správnou funkčnost modulu. Současně zajišťují, aby se v případě poškozených nebo podvržených dat skript nepokusil alokovat příliš velkou část paměti, což by mohlo zpomalit server nebo dokonce způsobit jeho pád.

AudioVideo modul – Testy v modulu AudioVideo nemají za cíl kontrolovat správný proces kódování a dekódování. Spoléhám se na testy vývojářů knihovny Libav. Účelem testů modulu je ověření bezchybného začlenění Libav do JavaScriptu. Případy testují:

- kódování obrazových a zvukových dat,
- dekódování obrazových a zvukových dat,
- transformaci obrazových a zvukových dat,
- extrakce informací o datech.

Testování funkcionality jednotlivých částí jsem prováděl průběžně. Chyby, které jsem odhalil jsem také okamžitě opravil, takže nemůžu předložit výsledky testování. Obecně nejkomplikovanější bylo získání vhodných testovacích dat. Přesný postup popisuji v kapitole Realizace. Testováním moduly vynikají nad mnoha ostatními z repozitáře – některé z nich nebyly podrobeny testování vůbec.

```
.

0 passing (28ms)
1 failing

1) kalkulačka sečte čísla:

    AssertionError: expected 9 to equal 10
    + expected - actual

    +10
    -9

    at Assertion.fail (C:\Use dules\should\lib\assertion.js:199:17)
    at Assertion.prop.(anonym er\AppDa\should\lib\assertion.js:64:14)
    at Context.<anonymous> (C node.js\lib\amf\test\test.js:6:22)
    at callFn (C:\Users\user\ cha\lib\runnable.js:251:21)
    at Test.Runnable.run (C:\ _modules\mocha\lib\runnable.js:244:7)
    at Runner.runTest (C:\Use dules\mocha\lib\runner.js:374:10)
    at C:\Users\user\AppData\ runner.js:452:12
    at next (C:\Users\user\Ap a\lib\runner.js:299:14)
    at C:\Users\user\AppData\ runner.js:309:7
    at next (C:\Users\user\Ap a\lib\runner.js:248:23)
    at Object._onImmediate (C de_modules\mocha\lib\runner.js:276:5)
    at processImmediate [as _immediateCallback] (timers.js:345:15)
```

Obrázek 6.1: Ukázka výstupu testovacího nástroje Mocha.js.

Kapitola 7

Instalační příručka

V této kapitole popisuji instrukce, jak na svůj vlastní server nainstalovat moduly RTMP, AMF a AudioVideo. Nutno zmínit, že pokud budeme moduly používat na webovém hostingu, musíme si vybrat takový, který Node.js podporuje. Většina běžných hostingů v České republice podporuje pouze jazyk PHP. Alternativou, avšak zpravidla dražší, je virtuální server nebo dedikovaný server, na kterém může administrátor instalovat libovolný software.

7.1 Instalace RTMP a AMF modulu

Pokud chceme používat pouze moduly RTMP a AMF, stačí nainstalovat Node.js podle běžných instrukcí. V sekci stažení oficiální stránky `http://nodejs.org/download/` je archiv s kompletním zdrojovým kódem a instalátory pro Linux, Windows, SunOS a MacOS. Dostupné jsou ve verzích pro 32bitový i 64bitový systém. Pokud používáme linuxovou distribuci Ubuntu nebo Debian, můžeme instalovat z repozitáře balíčků příkazem:

```
sudo apt-get install nodejs
```

Následně můžeme spouštět JavaScriptové soubory příkazem:

```
node script.js
```

Modul AMF ani RTMP se instalovat nemusí. Do svých zdrojových kódů je importujeme standardně – voláním metody require.

7.2 Instalace AudioVideo modulu

Moduly RTMP a AMF slouží pro tvorbu RTMP serveru a RTMP klienta, streamování a publikování multimédií, čtení a zápis FLV souborů, sdílené objekty

a vzdálené volání procedur. Ve většině aplikací si s těmito moduly vystačíme, proto jsou zcela úmyslně nezávislé na AudioVideo modulu, jehož instalace je komplikovanější. Část modulu je psána v jazyce C++ a proto musí být před spuštěním zkompilován pomocí třech kroků:

1. Instalace Node.js ze zdrojového kódu, tzn. stáhnout archiv dostupný na adrese `http://nodejs.org/download/` a spustit skript pro kompilaci.
2. Získat binární verze Libav, a to buď vlastní kompilací (popsanou na [32]) nebo stažení již zkompilovaných dynamických knihoven z oficiálního zdroje `http://builds.libav.org/`.
3. Instalace nástroje pro kompilaci C++ modulů. Programoval jsem ve vývojovém prostředí Microsoft Visual Studio (konkrétně ve verzi Microsoft Visual C++ 2010 Express). Na [26] je dostupný velmi pěkný návod, jak Visual Studio nastavit pro kompilaci Node.js modulů. Alternativní možností je multiplatformní nástroj node-gyp.

Nyní můžeme modul zkompilovat. Vygenerovaný soubor má příponu `.node`. Do JavaScriptových kódů jej importujeme funkcí `require` – stejně, jako by byl psán v JavaScriptu.

7.3 Instalace testovacích nástrojů

Pro testování zdrojového kódu je nutné mít nainstalovány testovací frameworky Mocha.js a Should.js. Instalaci obou můžeme provést velmi jednoduše z příkazové řádky:

1. Mocha.js

```
npm install -g mocha
```

2. Should.js

```
npm install -g should
```

Parametr `-g` (`global`) stáhne moduly do adresáře Node.js. Pokud parametr vynecháme, moduly se nainstalují do adresáře, ve kterém příkaz zavoláme. Testy spustíme jednoduše v adresářích `/src/amf/`, `/src/rtmp/` nebo `/src/audiovideo/` příkazem:

```
mocha
```

Další informace jsou dostupné na oficiální stránce `http://mochajs.org/`, resp. `https://github.com/tj/should.js`.

Kapitola 8

Ukázková aplikace

Ukázkové aplikace jsou internetové stránky, které pracují s multimediálním obsahem. Prokazují splnění cílů práce a správnou funkci softwaru.

Vytvořil jsem celkem 3 aplikace:

1. Player,
2. Publisher,
3. Video chat.

Všechny aplikace se nachází na doprovodném CD v adresáři `/src/app/`. Pro správnou funkci je nutné mít nainstalovaný Node.js a modul AudioVideo podle instrukcí v kapitole Instalační příručka.

8.1 Player

Aplikace Player demonstruje přehrávání video souborů, tedy to, co bychom od streamovacího serveru čekali v první řadě. Od moderního přehrávače nekompromisně vyžadujeme funkce:

1. Přehrávač zachycuje screenshot 8.1a. První, co člověka trkne do oka je černé pozadí. Uživatelé očekávají od přehrávače, že zobrazí ještě před spuštěním přehrávání snímek z videa. První funkce, kterou streamovací server nabízí je **náhled** na pozadí (viz. obrázek 8.1b).

2. Uživatel přehrává video, ale jak už to chodí, nemá dost trpělivosti na sledování celého dvouhodinového záznamu. Chtěl by se podívat jen na zajímavé scény. Druhou funkcí je **seekování**. Uživatel může ve videu skákat na libovolné časy. Streamovací server modulu RTMP dokonce podporuje tzv. *smart seeking* [33]. Je to vylepšení Flash Playeru 10.1 a novějších, které umožňuje skákat ve videu načteném v paměti. Pokud už má Flash Player v bufferu načtený čas, který chce uživatel zobrazit,

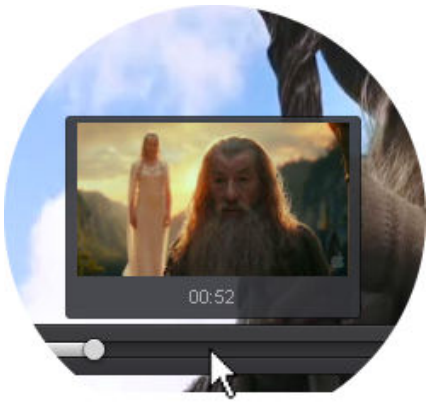
(a) bez náhledu (b) s náhledem

Obrázek 8.1: Přehrávač ukázkové aplikace.

může rovnou začít přehrávat. Starší verze toto neuměly, třebaže uživatel chtěl skočit na místo, které už bylo načteno, Flash Player stejně vymazal buffer a vyžádal si po serveru, aby mu poslal video od požadované pozice. *Smart seeking* ani zdaleka nepodporují všechny RTMP servery.

3. Skákání ve videu je pomalé! Přestože se streamovací servery snaží seč můžou, načtení nové pozice často trvá i několik sekund. Nevyhýbá se to ani službě YouTube. Aby uživatel nemusel skákat tak často, měl by mu přehrávač nejprve ukázat náhled – další podporovanou funkcí jsou **miniatury** na časové ose (viz obrázek 8.2)

4. Video někdy nedokáže říci vše, někdy potřebuje komentář. Streamovací server umí do videa vkládat **titulky**. Uživatel je může během sledování videa kdykoliv zapnout nebo vypnout, dokonce může zobrazit titulky v jiném jazyce. Titulky mohou být uloženy v souboru současně s videem nebo je může server generovat online. Používám slovo *titulky*, protože jsou snadno představitelné, ale přesněji řečeno to mohou být libovolná strukturovaná data (např. reklama, komentáře, obrázky apod.).

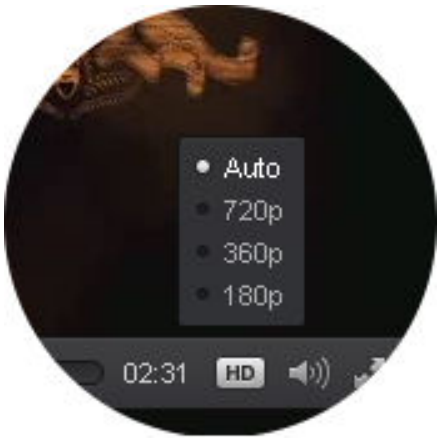


Obrázek 8.2: Miniatury na časové ose.

5. Poslední dosud nezmíněnou funkcí je **změna kvality** videa. Na obrázku 8.4 je menu přehrávače, na kterém si uživatel může zvolit kvalitu. Pokud uživatel nechá automatickou volbu, kvalita se přizpůsobí lince, aby se video přehrávalo plynule.

Zcela jsem opomenul různé efekty a tlačítka pro ovládání přehrávače. Server na jejich funkci nemá vliv, je to jen otázkou propracovanosti Flash skriptu. Pro přehrávání jsem použil volně dostupný Flash přehrávač JW Player, protože nabízí všechny zmíněné funkce, které bych musel jinak programovat. Stejně tak bych mohl použít Flowplayer nebo kterýkoliv jiný přehrávač. Tato práce se zabývá streamováním a publikováním dat, nikoliv prezentováním uživateli.

Obrázek 8.3: Titulky ve videu.



Obrázek 8.4: Volba kvality videa.

8.2 Publisher

Druhá aplikace s názvem Publisher demonstruje publikování dat z webkamery a mikrofonu na server. JW Player, jak už jeho název napovídá, je pouze přehrávač, publikovat nedokáže. Vytvořil jsem proto jednoduchou Flash aplikací, která zachycuje data z webkamery a mikrofonu a publikuje je na server.

Vše zajímavé se děje na straně serveru. Server videa ukládá s příponou `.flv` do adresáře `movies`. Video se neukládá pouze jako jeden soubor, nýbrž rovnou tři soubory v různých kvalitách. Původní verze uživatele, tedy ta s nejvyšší kvalitou, se uloží do souboru končícího na `high`. Celý název souboru je `movie.high.flv`. Verze s trochu nižší kvalitou (a současně menším datovým tokem) se uloží jako `medium`. Video v nejhorší kvalitě je pojmenováno `movie.low.flv`. Video se ukládá v různých verzích, aby mohl přehrávač měnit kvalitu při přehrávání. Server používá pro změnu kvality třídy z mo-

dulu AudioVideo: VideoMessageDecoder a VideoMessageEncoder, resp. AudioMessageDecoder a AudioMessageEncoder.

Při pohybu myší nad časovou osou přehrávače se zobrazují miniatury snímků. Vygenerovat obrázek ve chvíli, kdy si ho chce uživatel zobrazit by bylo příliš pomalé. Správnou cestou je obrázky vytvořit dopředu a uložit do souboru. Prohlížeč uživatele je v pravou chvíli během okamžiku stáhne z obyčejného webového serveru. Miniatury generuje aplikace každou sekundu. Dekóduje publikovaný obraz, zmenší jej a uloží do složky `previews`. Do složky s videem také vygeneruje náhled – hned první snímek, který je zachycen z webkamery. Náhled si zachovává rozlišení, na rozdíl od miniatur.

Aby mohl přehrávač skákat ve videu, musí vědět, jak je video dlouhé. Pochopitelně velikost videa se server dozví teprve ve chvíli, kdy jej uživatel celé nahraje, nikoliv když na začátek souboru zapisuje metadata. V kapitole Návrh popisuji trik, jakým server zpětně doplňuje délku videa do metadat na začátku souboru.

Se seekováním se váže druhý problém a tím je výpočet správné pozice pro skok. Pokud uživatel chce zobrazit konkrétní čas, nemůže streamovací server jen tak skočit někam do souboru a doufat, že tam přečte správná data. Této problematice se věnuji v části Návrh .meta souborů. V souboru je možné skákat pouze na pozice klíčových video snímků. Zjistit se dají jen sekvenčním procházením souboru od začátku do konce. Není nic snazšího, než si je zapamatovat při ukládání souboru. Jakmile je celé video nahráno, adresy klíčových snímků a další informace se uloží do souboru s příponou `.meta`. Při seekování server nemusí sekvenčně procházet celé video, které může mít i více než desítky nebo stovky MB, ale pouze načte malý soubor `movie.meta`.

8.3 Video chat

Aplikace Video chat slouží k video konferencím mezi uživateli. Ukazuje, jak server pracuje s živými přenosy. Přestože se to může zdát jako nejsložitější část, opak je pravdou. Přehrávání živého streamu je daleko jednodušší než přehrávání videa. Uživatelé mohou video pozastavit, skákat na jiné pozice, měnit kvalitu atd. Živé vysílání nic z toho neumí.

Pro tuto aplikaci jsem nemusel programovat žádné Flash skripty, složil jsem ji z Publisheru a Playeru. JW Player, který používám pro přehrávání videí v aplikaci Player dokáže přehrávat také *live stream*. Pro publikování dat z webkamery zase používám Flash skript Publisheru. Stránka Video chatu je rozdělena vertikálně na dvě poloviny. V pravé části návštěvník sleduje prostřednictvím Playeru obraz jiného uživatele a v levé polovině je Publisher, na kterém vidí sám sebe, jak je ve video konferenčních softwarech obvyklé.

Jakmile uživatel spustí kameru, data se začnou publikovat na server. Dokud je uživatel na stránce, může se k němu kdykoliv připojit jiný uživatel.

8.4 Shrnutí

Software, který jsem vytvořil, dokáže svými funkcemi konkurovat i největší video službě YouTube. Server nabízí tytéž funkce a navíc je libovolně rozšiřitelný. Přehrávání videí je např. možné zpoplatnit nebo omezit pouze pro přihlášené uživatele. V mnoha projektech, které se dosud musely spoléhat na externí službu, může sloužit k práci s multimediálním obsahem. Streamování a publikování je jen jedna z mnoha vhodných oblastí. Stejně dobře může sloužit k úpravě videí nebo aplikací obrazových a zvukových filtrů. Samotný modul AudioVideo je dost zajímavý na tvorbu několika vlastních aplikací.

Závěr

Hlavním cílem této práce bylo prozkoumat možnosti, jak vkládat multimediální obsah do internetových stránek. Technologie, které je možné používat na stránkách určuje internetový prohlížeč. Analýzu jsem proto začal podporovanými technologiemi prohlížečů. Bezpochyby nejvhodnější volbou je hojně podporovaný doplňek prohlížečů Flash Player. Jako jediný dokáže publikovat data z webkamery a mikrofonu uživatele na server. Ostatní technologie umí pouze video přehrávat. Prozatím to nedokáže ani HTML5 – největší favorit budoucnosti.

Flash komunikuje se serverem prostřednictvím síťového protokolu RTMP. Jak se ukázalo, společnost Adobe, která specifikaci publikovala, se dopustila v některých oblastech nepřesností, některé zcela opomíjí. Studium specifikace jsem doplnil analýzou RTMP serverů s otevřeným zdrojovým kódem, avšak neoficiální projekty protokol RTMP neimplementovaly správně. Komponenty Flash Playeru s nimi neumí pracovat, chovají se nestandardně. Abych si byl jistý skutečně správnou implementací, obrátil jsem se na zaručeně správný zdroj. Společnost Adobe vyvíjí oficiální streamovací server Adobe Media Server. Protože jeho zdrojové kódy nejsou veřejné, musel jsem správnou podobu protokolu zjistit metodami reverzního inženýrství.

Dle zadání práce jsem vyvíjel pro JavaScriptovou platformu Node.js. Práci jsem rozdělil do třech modulů. První modul implementuje protokol RTMP – vytváří server, streamuje a publikuje data, pracuje se soubory. Druhý modul slouží k serializaci a deserializaci AMF formátu, který se používá jako součást RTMP. Samotné tvorbě modulů předcházela rešeršní analýza existujících řešení. Ve veřejném repozitáři Node.js je spousta modulů související s problematikou. Jako základ práce jsem zvolil node-rtmpapi a node-amfutils, nicméně nakonec se ukázalo snazší vytvořit nové moduly než do nich doplnit funkcionalitu a odstranit všechny jejich chyby.

Video ani zvuk se v protokolu RTMP neposílá v surové podobě. Datový tok by byl příliš velký a přehrávání trhané. Data jsou komprimována kodeky. Pro rekonstrukci původní podoby dat je potřeba rozumět algoritmům kodeků.

Třetí modul, který jsem vytvořil, nese název AudioVideo. Jeho prvořadým úkolem je kódování a dekódování dat. Flash rozumí bezmála 20 kodekům. Není rozumné a ani technicky možné v rozsahu diplomové práce je všechny implementovat, proto modul AudioVideo využívá pro podporu kodeků knihovnu Libav, která se na tento úkol specializuje.

Výstupem práce jsou tedy tři moduly. Jak je správně použít, zejména modul RTMP, popisuji v kapitole Použití serveru. Volba kodeků a parametrů kódování ovlivňuje obrovským způsobem kvalitu videa a zvuku. V kapitole objasňuji všechna možná nastavení, včetně jejich vlivu na kvalitu a velikost dat a s tím související zatížení streamovacího serveru.

Správnou funkci modulů dokládají ukázkové aplikace. První aplikace, s názvem Player, přehrává video a vše s tím související. Podporuje všechny funkce známé z YouTube a dalších streamovacích služeb. Videa mohou být vložena administrátorem nebo nahrána uživateli pomocí druhé aplikace, aplikace Publisher. Jak snadno lze pořádat video konference mezi uživateli ukazuje poslední aplikace – Video chat.

Práce popisuje tvorbu softwarového díla od rozsáhlé analýzy, přes návrh, implementaci až po neméně důležité testování. Zadání a vytyčené cíle byly splněny. Vznikl funkční software, jehož schopnosti demonstrují ukázkové aplikace.

Literatura

[1] Pfeiffer, S.: *HTML5 - audio a video: kompletní průvodce. [online].* Brno: Zoner Press, první vydání, 2011, ISBN 978-80-7413-147-9.

[2] Sharovatov, V.: *HTML5 video tag and Internet Explorer. [online].* 2009, [cit. 2015-03-01]. Dostupné z: `https://sharovatov.wordpress.com/2009/06/05/html5-video-tag-and-internet-explorer/?`

[3] Kyrnin, J.: *The bgsound Element - Play background music on your page - Internet Explorer. [online].* [cit. 2015-03-01]. Dostupné z: `http://webdesign.about.com/od/htmltags/p/bltags_bgsound.htm`

[4] w3schools.com: *HTML <embed> Tag – Complete HTML Reference. [online].* [cit. 2015-03-01]. Dostupné z: `http://www.w3schools.com/tags/tag_embed.asp`

[5] w3schools.com: *HTML <object> Tag – Complete HTML Reference. [online].* [cit. 2015-03-01]. Dostupné z: `http://www.w3schools.com/tags/tag_object.asp`

[6] w3schools.com: *HTML5 Video. [online].* [cit. 2015-03-01]. Dostupné z: `http://www.w3schools.com/html/html5_video.asp`

[7] w3schools.com: *HTML5 Audio. [online].* [cit. 2015-03-01]. Dostupné z: `http://www.w3schools.com/html/html5_audio.asp`

[8] Adobe: *ACTIONSCRIPT 3.0 – Příručka pro vývojáře. [online].* [cit. 2015-03-01]. Dostupné z: `http://help.adobe.com/cs_CZ/as3/dev/as3_devguide.pdf`

[9] Kopta, M.: *Podpora pro Flash, JavaScript a cookies. [online].* 2004, [cit. 2015-03-01]. Dostupné z: `http://www.lupa.cz/clanky/podpora-pro-flash-javascript-a-cookies/`

[10] Adobe: *Adobe's Real Time Messaging Protocol. [online]*. 2012, [cit. 2015-03-01]. Dostupné z: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

[11] Adobe: *Adobe Media Server 5.0.7 Technical Overview. [online]*. [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/en_US/adobemediaserver/techoverview/

[12] Adobe: *Action Message Format – AMF 0. [online]*. [cit. 2015-03-01]. Dostupné z: http://download.macromedia.com/pub/labs/amf/amf0_spec_121207.pdf

[13] Adobe: *Action Message Format – AMF 3. [online]*. [cit. 2015-03-01]. Dostupné z: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/amf/pdf/amf-file-format-spec.pdf

[14] Node.js: *Node.js*. [cit. 2015-03-01]. Dostupné z: http://nodejs.org/

[15] La plaga tux: *Chrome V8 – Google Developers. [online]*. 2013, [cit. 2015-03-01]. Dostupné z: http://plagatux.es/2011/07/using-libav-library/

[16] Joyent, Inc.: *Node.js v0.12.0 Manual and Documentation. [online]*. [cit. 2015-03-01]. Dostupné z: http://nodejs.org/api/all.html

[17] Joyent, Inc.: *Node.js v0.12.0 Addons. [online]*. [cit. 2015-03-01]. Dostupné z: http://nodejs.org/api/addons.html

[18] libav.org: *About Libav. [online]*. [cit. 2015-03-01]. Dostupné z: https://libav.org/about.html

[19] Larabel, M.: *A Group Of FFmpeg Developers Just Forked As Libav. [online]*. 2011, [cit. 2015-03-01]. Dostupné z: http://www.phoronix.com/scan.php?page=news_item&px=OTIwNw

[20] La plaga tux: *Using libav library. [online]*. 2013, [cit. 2015-03-01]. Dostupné z: http://plagatux.es/2011/07/using-libav-library/

[21] Adobe: *Adobe Flash Video File Format Specification. [online]*. 2010, [cit. 2015-03-01]. Dostupné z: http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf

[22] Red5: *Flash Video (FLV) – About the Flash Video file format. [online]*. 2013, [cit. 2015-03-01]. Dostupné z: https://code.google.com/p/red5/wiki/FLV

[23] Mendelova univerzita v Brně: *Komprimace multimediálních dat. [online]*. [cit. 2015-03-01]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7020

[24] Mozilla: *JSON - Summary. [online].* [cit. 2015-03-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

[25] Drew, A.: *Information that is missing from the Adobe RTMP Specification. [online].* 2009, [cit. 2015-03-01]. Dostupné z: http://japanesesoapbox.blogspot.cz/2009/11/information-that-is-missing-from-adobe.html

[26] Romaniello, J. F.: *Writing your first native module for node.js on Windows. [online].* 2012, [cit. 2015-03-01]. Dostupné z: http://joseoncode.com/2012/04/10/writing-your-first-native-module-for-node-dot-js-on-windows/

[27] Adobe: *Camera – Adobe ActionScript 3 (AS3) API Reference. [online].* [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/cs_CZ/FlashPlatform/reference/actionscript/3/flash/media/Camera.html

[28] Adobe: *Adobe Flash Platform * Práce s kamerami. [online].* [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/cs_CZ/as3/dev/WSfffb011ac560372f3fa68e8912e3ab6b8cb-8000.html

[29] Adobe: *Microphone - Adobe ActionScript 3 (AS3) API Reference. [online].* [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/cs_CZ/FlashPlatform/reference/actionscript/3/flash/media/Microphone.html

[30] Adobe: *Adobe Flash Platform * Záznam zvukového vstupu. [online].* [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/cs_CZ/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7d1d.html

[31] FFmpeg: *Supported media types in formats. [online].* 2013, [cit. 2015-03-01]. Dostupné z: https://trac.ffmpeg.org/wiki/SupportedMediaTypesInFormats

[32] LibAV: *Platform Specific information. [online].* [cit. 2015-03-01]. Dostupné z: https://libav.org/platform.html

[33] Adobe: *Flash Media Server 4.5 – Smart Seeking. [online].* [cit. 2015-03-01]. Dostupné z: http://help.adobe.com/en_US/flashmediaserver/devguide/WSd391de4d9c7bd609-569139412a3743e78e-8000.html

Příloha **A**

Seznam použitých zkratek

AMF Action Message Format

AMS Adobe Media Server

API Application Programming Interface

FLV Flash Video

GOP Group Of Pictures

HD High definition

HLS HTTP Live Streaming

HTML HyperText Markup Language

JS JavaScript

NPM Node Package Manager

PNG Portable Network Graphics

RMI Remote Method Invocation

RTMFP Real Time Media Flow Protocol

RTMP Real Time Messaging Protocol

SO SharedObject

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

VLC VideoLAN

VOD Video on demand

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` — stručný popis obsahu CD
- `src` — zdrojové kódy
  - `amf` — zdrojové kódy AMF modulu
    - `test` — testy AMF modulu
  - `rtmp` — zdrojové kódy RTMP modulu
    - `test` — testy RTMP modulu
  - `audiovideo` — zdrojové kódy AudioVideo modulu
    - `test` — testy AudioVideo modulu
  - `app` — zdrojové kódy ukázkových aplikací
- `doc` — dokumentace zdrojových kódů
  - `amf` — dokumentace AMF modulu
  - `rtmp` — dokumentace RTMP modulu
  - `audiovideo` — dokumentace AudioVideo modulu
  - `generate.bat` — generátor dokumentace
- `thesis` — zdrojová forma práce ve formátu LaTeX
- `text` — text práce
  - `thesis.pdf` — text práce ve formátu PDF
  - `thesis.ps` — text práce ve formátu PS